Conception en vue de test de convertisseurs de signal analogique-numérique de type pipeline.

Asma Laraba

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Nano-électronique et Nano-technologies**

Arrêté ministériel : 7 août 2006

Présentée par

Asma Laraba

Thèse dirigée par **Salvador Mir** et
Co-encadré par **Haralampos-G. Stratigopoulos**

préparée au sein du **Laboratoire TIMA**
dans **l'École Doctorale Électronique, Électrotechnique, Automatique et Traitement du Signal (E.E.A.T.S)**

Conception en vue du test des Convertisseurs Analogique-Numérique de type pipeline

Thèse soutenue publiquement le **20 Septembre 2013**,
devant le jury composé de :

M. Patrick Loumeau
Professeur, Telecom ParisTech, Président
Mme. Adoración Rueda
Professeur, Universidad de Sevilla (Espagne), Rapporteur
M. Hans Kerkhoff
Professeur, University of Twente (Pays-Bas), Rapporteur
M. Olivier Rossetto
Maître de Conférences, Université Joseph Fourier, Examinateur
M. Hervé Naudet
Ingénieur Senior, STMicroelectronics, Invité
M. Salvador Mir
Directeur de recherche, CNRS, Directeur de thèse
M. Haralampos-G Stratigopoulos
Chargé de recherche, CNRS, Co-encadrant de thèse

ISBN : **978-2-11-129184-3**

A ma mère, أمِّي

A mon père, أبِي

Remerciements

Ces lignes, les dernières écrites, marquent la fin de trois belles années de thèse passées à Grenoble entre le laboratoire Tima et STMicroelectronics. Pour commencer, je tiens à remercier infiniment mes encadrants, Salvador et Haralampos, grâce à qui cette thèse a été une réussite. Je vous remercie d'abord pour m'avoir fait confiance dès mon stage de Master, de m'avoir guidé pour l'aboutissement de ce projet, je vous remercie pour votre temps, votre patience, vos conseils, votre bienveillance, surtout pour vos qualités humaines, qui, certainement, ont fait la différence. Votre attitude envers vos doctorants est sans doute introuvable ailleurs! Je vous suis très reconnaissante pour beaucoup de choses, je ne finirai pas de vous remercier, et je passe aux autres...

Je remercie sincèrement Hervé Naudet, leader de l'équipe 'DFT and test engineering' de ST, pour avoir accepté de me confier la résolution de ce problème industriel, d'avoir initié cet intéressant projet et d'avoir été présent et disponible tout au long des trois années de thèse. Je remercie aussi les ingénieurs de l'équipe 'High-Speed ADCs' de ST qui m'ont supporté et qui m'ont appris beaucoup de choses. Je pense à Gerard, Christophe, Marc, Hugo, Fouad, Sophie, Roger, et Fabien! Merci aussi pour votre bonne humeur et les pauses déjeuner toujours agréables!

Présents sur la dernière ligne droite, je voudrai remercier les membres du jury de thèse, qui m'ont honoré en acceptant de faire partie de ce jury, Patrick Loumeau, Hans Kerkhoff et Adoracion Rueda, surtout, pour avoir été venu de loin pour assister à la soutenance. Un merci spécial à Olivier Rossetto, le responsable de mon Master et mon ancien prof; et Hervé Naudet, qui, malgré la période de vacances, a pu lire et commenter mon rapport de thèse en direct d'une belle plage!

Les collègues et amis de Tima, je pense surtout à l'équipe déjeuner, mes deux chères Lyl et Giota, Haralampos, Martin, Thanasis, Guillaume et Manuel, merci pour ces pauses gourmandes toujours très agréables, la Bon'Heure et les quiches-salades me manquent déjà, un peu moins le RU! Je remercie aussi l'ancienne génération de l'équipe RMS pour leur précieux conseils, étant fraîchement arrivée au monde de la recherche. Je pense surtout à Matthieu, Rafik, Nourreddine, Ke, Louay, Hakim, Oussama et Fabio, et bien sûr Josué et Adrien, étant nouveaux doctorants aussi, pour le partage des quelques frustrations de début de thèse!

Un super grand merci à ma très chère famille! Mes parents surtout à qui je dois tout, mes deux superbes soeurs Ibtissem et Abir et mes frères bienveillants, Khaled et Amine. Je vous remercierai surtout en dehors de cette petite section! Je finis sans oublier ma très chère Yasmine et mon cher Lokmane.

Si j'avais manqué de mentionner des noms, je m'en excuse... Merci à tous ceux et celles que j'ai eu le plaisir et la chance de connaître ou d'avoir dans ma vie et qui ont contribué à cet aboutissement, de loin ou de près, d'une manière ou d'une autre.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction

During the last decades we have witnessed an enormous development in consumer electronics. What have been thought of as fiction 50 years ago is nowadays of the order of the usual and is within everybody's reach. The growing capabilities of electronic devices is strongly related to the increasing integration levels that are still keeping track of Moore's prediction. Processing speeds, memory capabilities, etc, are all improving at approximately exponential rates. From a product point of view, performances are getting better at the cost of increasing design complexity.

The typical design flow of a product is shown in Figure 1.1. After each major step, a validation is needed: first after layouting the design in the form of post-layout simulations. Second after fabricating the first test-chip, this step is referred to as characterization. And finally before shipping the chips to the customers, this step is referred to as production test.

Depending on the type of circuit, the production test strategy is different. The test of a digital circuit (microprocessors, logic blocks, memories...) consists in injecting test vectors that permit the detection of the presence of faults. Detecting faulty circuits is a simple comparison between test results and expected responses. The generation of test vectors is conducted during the design phase of the circuit. The developpment of CAD tools have lead to increasing automation of these procedures. On the other hand, testing analog (operational amplifiers, active or passive filters, comparators, voltage regulators, analog mixers, voltage or current references...) and mixed signal circuits (Analog-to-Digital Converters, Digital-to-Analog Converters, phase locked loops, programmable gain amplifiers...) is based on the verification of the design specifications. Its development consists in setting an adequate test environment (low noise, low interference,...etc) and choosing the adequate signal generators.

Analog and mixed-signal test is getting more and more challenging as the products performances and complexities are increasing. Analog-to-Digital Converters (ADCs) are the most common mixed-signal devices and are present in every mixed-signal system. For ADCs, increasing performance usually translates in higher resolution. Differential Non Linearity (DNL) and Integral Non Linearity (INL) are the two main static

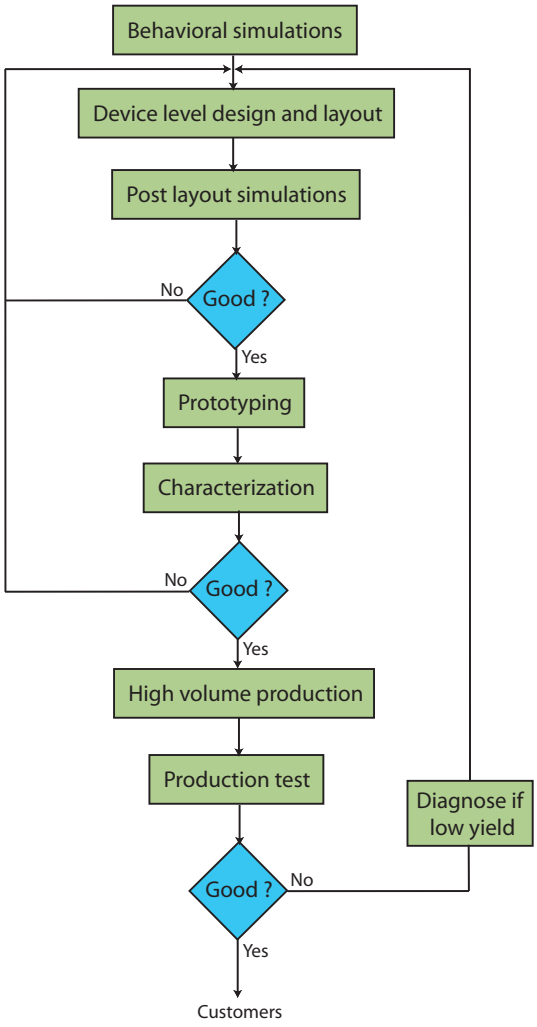


Figure 1.1: Design flow of an integrated circuit.

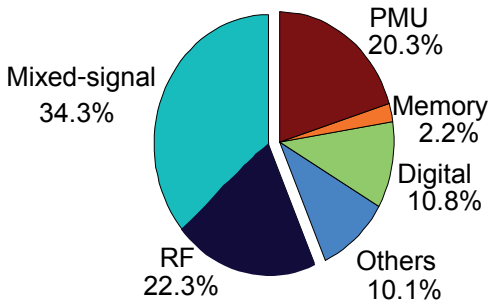


Figure 1.2: Test times per circuitry type.

performances that are measured during production testing of ADCs. In a classic testing scheme, a saturated sine-wave or ramp is applied to the ADC and the number of occurrences of each code is obtained to construct the histogram from which DNL and INL can be readily calculated.

This standard approach requires the collection of a large volume of data because each code needs to be traversed many times to average noise. The volume of data that needs to be collected increases exponentially with the resolution of the ADC under test. The test time becomes prohibitively large for high-resolution ADCs and in fact although mixed-signal circuits occupy an area less than 5% in a modern SoC, testing the mixed-signal functions takes up to 30% of the total test time. Figure 1.2 shows the production test time for each circuit type in a typical SoC [8].

1.2 Motivations and research contribution

Increasing test time translates to increasing test costs. Nowadays, the static test of high resolution ADCs is addressed with the same approaches used for low resolution ADCs and the test time is not reasonable regarding the silicon area that is tested or the test time of the other circuits. There is no doubt that the time has come to search for and find a replacement of the actual standard testing techniques. If not, the test cost will dominate the cost of the fabrication of an integrated circuit and this will inevitably stand in the way of the continuous developments in consumer electronics.

This thesis was conducted in the frame of a project between Tima Laboratory and STMicroelectronics. The goal was to find a new way of testing pipeline ADCs to cut down their ever increasing static test time.

Pipeline ADCs offer a good compromise between speed, resolution, and power consumption. It is the most attractive architecture for resolutions higher than 8 bits and sampling rates in the range of tens to hundreds of MHz. This is well-suited for a wide range of applications such as high-definition digital video (HDTV, HDMI..), cable modems, fast ethernet, ultrasonic medical imaging, digital receivers, etc.

I studied and analysed deeply the design, architecture and functionality of pipeline ADCs based on which I proposed a very efficient reduced code linearity test technique. Reduced code testing can be applied to ADCs that, by virtue of their operation, have groups of output codes which have the same width. Thus, instead of considering all

the codes in the testing procedure, we consider measuring only one code width out of each group, thus reducing significantly the static test time.

1.3 Thesis overview

After this brief introduction Chapter, the second Chapter will introduce ADC linearity test principles and the state of the art of the alternative linearity testing techniques. In the third Chapter, an overview of pipeline ADCs is presented and the redundancy in the static testing of pipeline ADCs is explained. After showing the limitations of the existing reduced code testing technique of pipeline ADCs, we propose a new technique that overcomes the shortcomings of the existing technique and permits applying reduced code testing efficiently. The proposed technique is sensitive to noise and thus in the fourth Chapter we shed light on another feature in pipeline ADCs that can be exploited in order to cancel out the inaccuracies that are due to the presence of noise. In the fifth Chapter we show experimental results obtained on an 11-bit, 55nm pipeline ADC from STMicroelectronics. Finally, the sixth Chapter provides conclusions and discusses the perspectives of this work.

Chapter 2

State-of-the-art of ADC linearity testing

2.1 ADC linearity test principles

2.1.1 Definitions

ADCs sample and quantize continuous analog input signals and generate outputs with discrete values at discrete times. Figure 2.1 shows the ideal transfer characteristic of an ideal 3-bit ADC. In the case of an ideal ADC all the codes have equal widths and the curve which passes through the middle points of the steps is a straight line. The ideal code width is equal to 1 LSB (Least Significant Bit) defined as follow:

$$LSB = \frac{T_{N-1} - T_1}{N - 2} \tag{2.1}$$

where N is the number of codes of an n-bit ADC ($2^n = N$) and T_i is the transition voltage of code i. Notice that the first and last codes of an ADC do not have a defined width.

Static errors reflect a non-ideal spacing of the code transition levels. The most important static specifications are the Integral Non-Linearity (INL) and the Differential Non-Linearity (DNL).

The DNL of a certain code is the deviation of its width from the ideal one LSB width (cf. Figure 2.2). This is expressed as:

$$DNL(i) = \frac{T_{i+1} - T_i}{LSB} - 1, i \in [1, N - 2] \tag{2.2}$$

INL is a measure of the deviation of the transfer characteristic from a straight line (cf. Figure 2.2). A conservative measure is to use the endpoints of the ADC's transfer characteristic to define the straight line. An alternative definition is to find the best-fit straight line such that the maximum difference is minimized. The INL can be obtained by integrating the DNL curve:

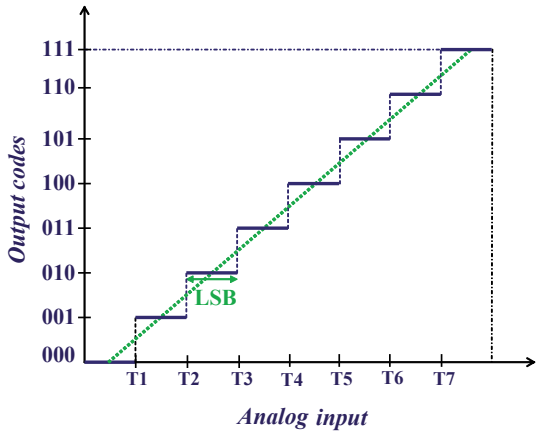


Figure 2.1: Ideal ADC transfer characteristic.

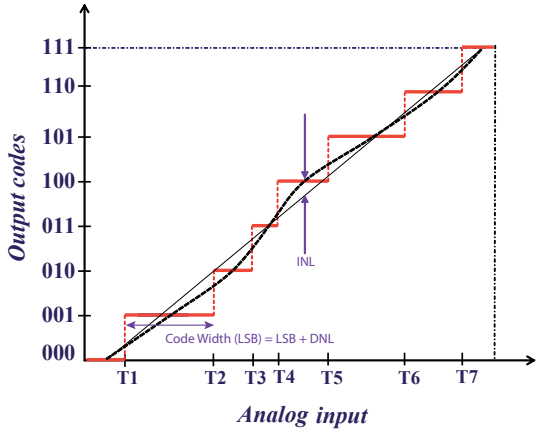


Figure 2.2: DNL and INL illustration on a non-ideal ADC transfer characteristic.

$$INL(i) = \sum_{k=1}^{i-1} DNL(k) \tag{2.3}$$

Offset and gain errors can be determined from the transfer characteristic easily by looking at the deviations of the first and last transition levels of the actual characteristic from the ideal ones. In most applications, these two errors do not affect the linearity performance of the system and can be simply compensated.

2.1.2 Linearity test

ADC linearity testing consists in extracting the above described static performances. An analog stimulus signal is applied to the ADC under test and the digital outputs of the ADC are monitored or post processed to extract the static performances of the ADC. There are two test methods usually used: the histogram method and the

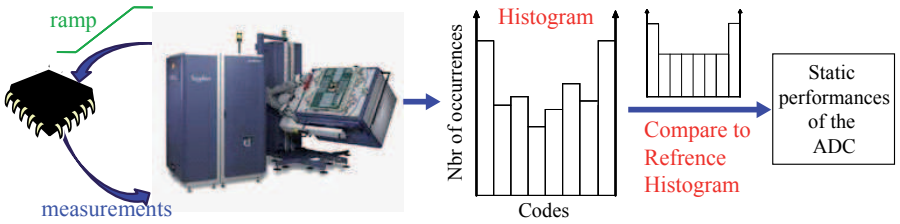


Figure 2.3: Ramp histogram test principle.

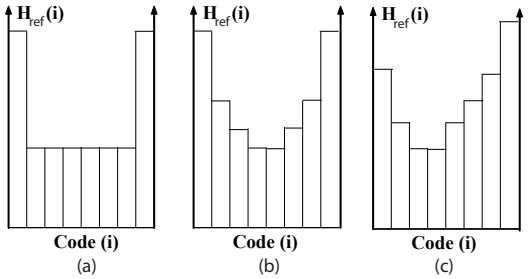


Figure 2.4: (a) Reference histogram of a linear ramp, (b) reference histogram of a symetrical sinusoidal signal, and (c) Reference histogram of a sinusoidal signal with offset.

servo-loop method (known also as the feedback loop method).

Histogram test

In the histogram approach, a histogram of code occurrences is obtained by applying an input signal with a known distribution over the ADC input range (Figure 7.3). The number of occurrences of each code is plotted as a histogram. For example, if the input signal is a linear ramp, each code should be hit the same number of times ideally. This would only be true for a perfectly linear ADC. The codes which are hit more often are wider codes. After a sufficiently large number of samples, the distribution of the output histogram compared to the known distribution of the input permits extracting the static performances of the ADC.

A histogram test can either be performed with a linear ramp or with a sine-wave. The linear ramp histogram testing is significantly faster than sine-wave histogram testing because a much smaller number of samples need to be collected for a desired accuracy. Ramp histogram testing requires a very accurate and linear ramp. If the ramp has non-linearity errors, they will be translated into equivalent errors in the measured ADC transfer characteristic. The input ramp is usually generated by a high-resolution DAC or an arbitrary waveform generator with suitable linearity. Note that the ramp amplitude should be sufficient to slightly overdrive the ADC.

The total number of the collected histogram samples is very important to the precision of the measured static performances. Increasing the number of samples decreases the uncertainty in the measurements and the effects of the noisy environment.

Another widely used stimulus in histogram test is the sine wave, which is easier to obtain than a linear ramp signal. During the test, a pure sine wave of amplitude sufficient to slightly overdrive the ADC is fed to the ADC under test. The frequency of the sine wave and the length of the data to be collected must be carefully selected based on the sampling frequency of the ADC if coherent sampling is used (non-coherent sampling requires collection of a larger number of samples). The transition levels can then be calculated from the histogram data and the known distribution of the sine-wave samples [9],[10], [11]. Since the distribution of the sine-wave samples is not uniform, the calculation involves trigonometric calculations. Also, to make sure there are sufficient samples at the center of the input range, a larger number of samples are needed for testing than with a ramp signal.

The reference histogram for a linear ramp is easy to obtain: all ADC codes except the first and the last codes have the same probability to appear and the same reference histogram $H_{ref}(i)$ (Figure 2.4(a)). In the case of a linear ramp input signal:

$$H_{ref}(0) = H_{ref}(2^n - 1) = N_T * \left(\frac{2^{n-1} * (A_{in} - FS) + FS}{2^n * A_{in}} \right) \tag{2.4}$$

$$H_{ref}(i) = N_T * \frac{FS}{2^n * A_{in}} \; ; i \in [1, 2^n - 2] \tag{2.5}$$

where $H_{ref}(i)$ is the ideal number of occurrences of code i, N_T is the total number of samples, n is resolution of the ADC, FS is the full scale of the ADC and A_{in} is the peak to peak amplitude of the ramp. If $H(i)$ is the number of occurences of code i then:

$$DNL(i) = \frac{H(i) - H_{ref}(i)}{H_{ref}(i)} LSB \tag{2.6}$$

For a sinusoidal histogram test, the reference histogram for each code is different (Figure 2.4(b)). In the case when the mid-range of the sinusoidal signal perfectly matches the mid-range of the ADC under test, then:

$$H_{ref}(0) = H_{ref}(2^n - 1) = \frac{N_T}{\pi} * \left(arcsin \left[\left(\frac{1}{2^{n-1}} - 1 \right) * \frac{FS}{A_{in}} \right] - \frac{\pi}{2} \right) \tag{2.7}$$

$$H_{ref}(i) = \frac{N_T}{\pi} * \left(arcsin \left[\left(\frac{2 * i - 2^n}{2^n} \right) * \frac{FS}{A_{in}} \right] - arcsin \left[\left(\frac{2 * i - 2^n - 2}{2^n} \right) * \frac{FS}{A_{in}} \right] \right); \tag{2.8}$$

In practice, the symetry condition is rarely (never!) met. Actually, when the sinusoidal waveform is offsetted with respect to the mid-range of the ADC then the reference histograms of the first and last codes are not equal. The difference of occurrences of these extreme codes gives information about the amount of offset of the input signal. With this information we can calculate a reference histogram that is adequate to the stimulus (Figure 2.4(c)). For this purpose, we first need the calculate the offset V_o and amplitude A_o of the signal as seen by the ADC [10]:

$$V_o = \left(\frac{2^n}{2} - 1 \right) * \left(\frac{(cos(\pi * \frac{N2}{N})) - (cos(\pi * \frac{N1}{N}))}{(cos(\pi * \frac{N2}{N})) + (cos(\pi * \frac{N1}{N}))} \right) \tag{2.9}$$

$$A_o = \frac{(\frac{2^n}{2} - 1) - V_o}{cos(\pi * \frac{N1}{N})} \tag{2.10}$$

where $N1$ is the number of occurrences of the last code and $N2$ is the number of occurrences of the first code. In this way, the reference histogram for code i will be given by:

$$H_{ref}(i) = \frac{N_T}{\pi} * \left(arcsin\left[\frac{i - (\frac{2^n}{2} - 1) - V_o}{A_o}\right] - arcsin\left[\frac{i - (\frac{2^n}{2}) - V_o}{A_o}\right]\right); i \in [1, 2^n - 2] \tag{2.11}$$

Servo loop test

The principle of the servo-loop test technique is shown in Figure 7.4. In this approach, an input is applied to the ADC under test and the ADC outputs are compared to a reference code k, which specifies the code transition level $T(k)$ that needs to be determined.

If the ADC output is below the preset value, the input is raised by a certain amount. If the ADC output is equal to or above the desired value, the input is reduced by a certain amount. This process is repeated until the ADC input has settled to a stable average value [9, 12]. After the loop has settled, the input is a representation of the code transition level $T(k)$. Its value can then be either measured by a high-precision digitizer or computed from the known transfer function of the input source.

The analog input to the ADC can be generated by either a high-resolution DAC (Figure 7.4(a) or an analog integrator (Figure 7.4(b)). When using a DAC as source generator, the resolution of the DAC should be higher than the resolution of the ADC under test. This is due to the fact that the amplitude of the voltage steps generated by the DAC should be smaller than the LSB of the ADC under test (up to the desired precision of the test). The amplitude of the steps is also dependant on the noise level of the ADC. Detailed analysis of all these considerations can be found in [9],[13], [14] and [15].

2.2 Alternative linearity testing techniques

As the integration level increases and products become more complex, applying conventional testing methods with satisfactory testing performance and cost becomes increasingly difficult. Conventional linearity testing methods require either high-performance source signals or high-precision digitizers and this translates in using expensive Automatic Test Equipments (ATE). Reducing the static test time and test cost is an important challenge nowadays and many alternative test techniques for ADCs have been reported to date. We can classify them in two groups: external testing techniques or Design-For-Test techniques that propose methods to make the standard testing techniques less problematic and Built-In Self-Techniques where the purpose is to integrate on-chip the standard external test techniques. In the following we will give an overview of the most important contributions.

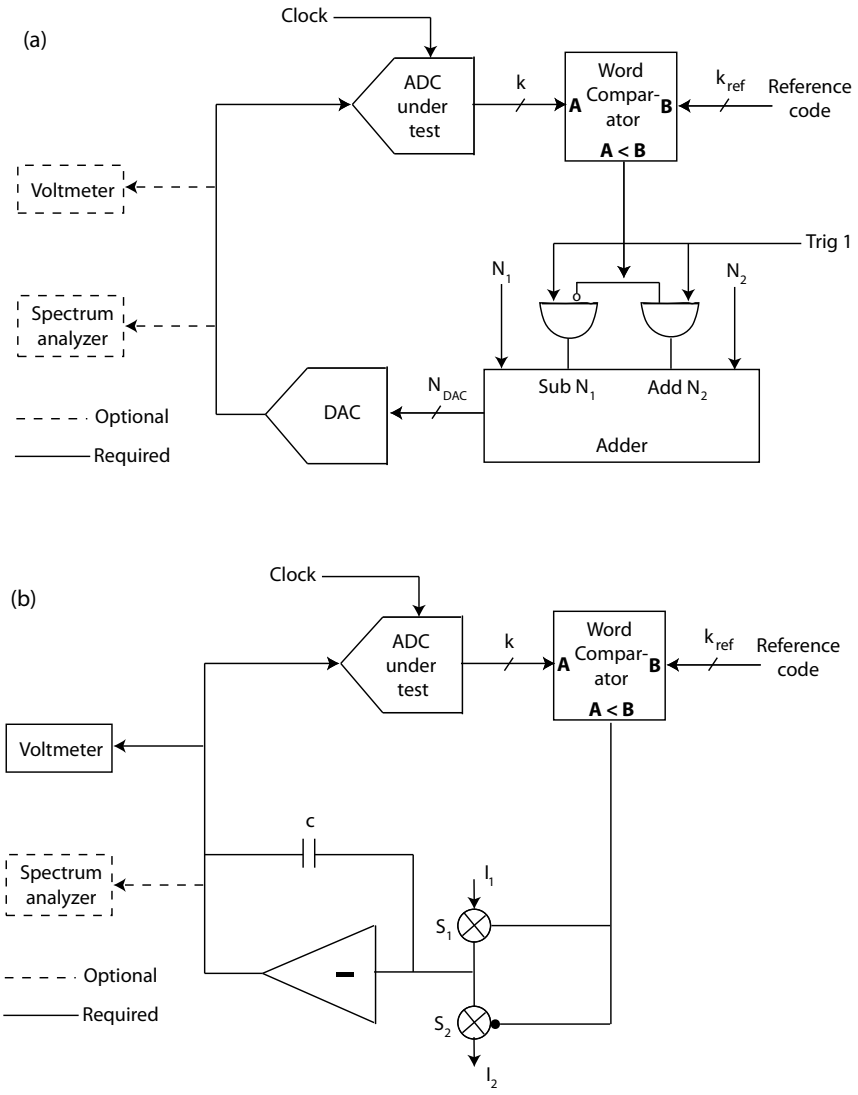


Figure 2.5: The servo-loop: (a) digital approach, and (b) analog approach.

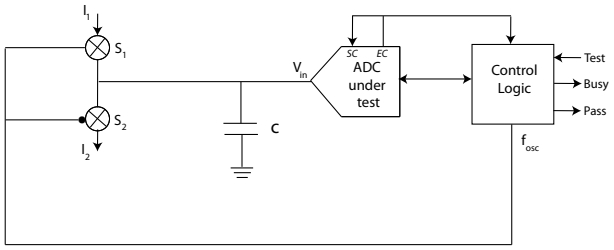


Figure 2.6: Oscillation BIST.

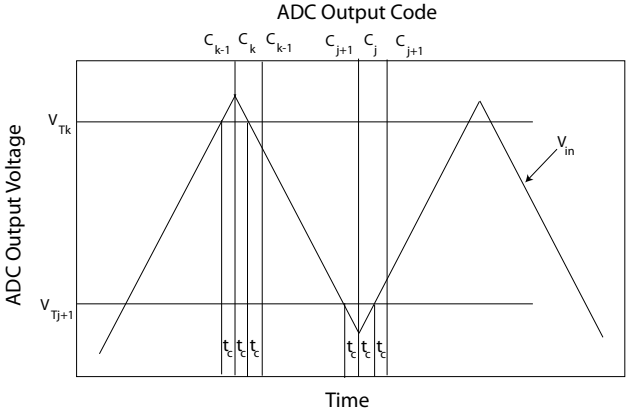


Figure 2.7: ADC input voltage oscillation between V_{Tk} and V_{Tj+1}.

2.2.1 Built-in Self-Test Techniques

These techniques are not ADC-specific and the goal is to either integrate the signal generation or the data-analysis on-chip, or both.

Oscillation BIST

Figure 2.6 shows the principle of the oscillation-based BIST technique [16]. It is inspired from the standard servo-loop technique, except that in an integrated scheme the system is forced to oscillate between two determined codes (Figure 2.7). The frequency of the oscillation depends on the conversion time and on the transition voltages of the two codes. First, the conversion time (t_c) is determined by forcing the ADC to oscillate between C_k and C_{k-1} (in this case, the ADC input is locked to V_{Tk}). In order to determine the DNL of code C_k, the ADC output is forced to oscillate between C_k and C_{k-2}. Thus, V_{in} will oscillate between V_{Tk} and V_{Tk-1}. The oscillation frequency is correlated to (V_{Tk}-V_{Tk-1}), which is the code width of C_k. The INL of code C_k can be obtained by accumulating the DNL errors of the preceding codes. Otherwise, the INL of code C_k can be evaluated directly by forcing the ADC output to oscillate between C_0 and C_{k+1}. Thus, V_{in} will oscillate between V_{T1} and V_{Tk+1}. By digitally measuring the oscillation frequency the static performances of the ADC can be deduced (the

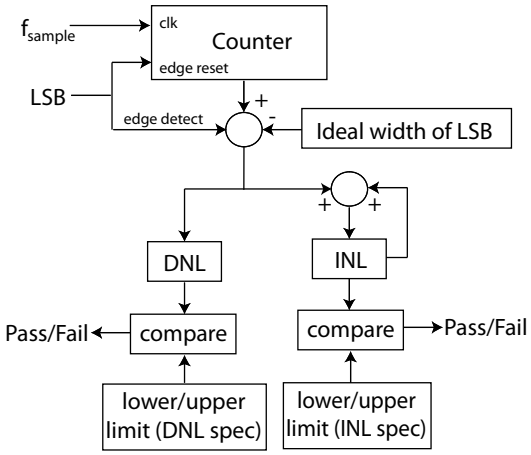


Figure 2.8: Counter based on-chip DNL and INL calculation [1].

dedicated equation for each static performance can be found in the paper). In [17] the same principle is further analysed and a digital circuit for accurately controlling the integration time is proposed. Unfortunately, no hardware results have been published for the oscillation BIST of ADCs. Also, in the published papers the analysis of the impact of noise on the accuracy of the technique has not been approached even though this is an important factor in such techniques.

Counter based on-chip DNL and INL calculation

Since the Least Significant Bit (LSB) of an ADC always transitions between two successive codes, the authors in [1] propose to observe the changes in the LSBs to extract the static performances of the ADC (Figure 2.8). This can be conducted under the assumption that the non-linearity errors are not very large. For the Most Significant Bits, this method only guarantees that the bits are consecutif. The linearity is calculated using measurements obtained with a clock and a counter. The counter starts counting when the LSB makes a transition. At the next transition the content of the counter is compared with an upper and a lower limit given by the DNL specifications of the ADC after which a pass/fail decision is made for the tested code. The INL of each code is determined by accumulating the DNLs of the preceding codes. The obtained INL value is compared with an upper and lower limit upon which a pass/fail decision is made. The clock jitter limits the precision of this technique and a good linearity of the input signal is needed. Also, in a real ADC the LSB transitions are usually very noisy and the authors do not discuss how to overcome this issue when using the proposed technique.

A similar idea is presented in [2] and [3]. Instead of calculating the number of clock cycles separating two successive transitions, the ADC output is compared to the output of a counter. The number of bits m of the counter is higher than the resolution n of the ADC, up to the desired precision of the measurement. Figure 2.9 shows an example in the case of a 2-bit ADC and a 5-bit counter. INL is evaluated by comparing the n-bits

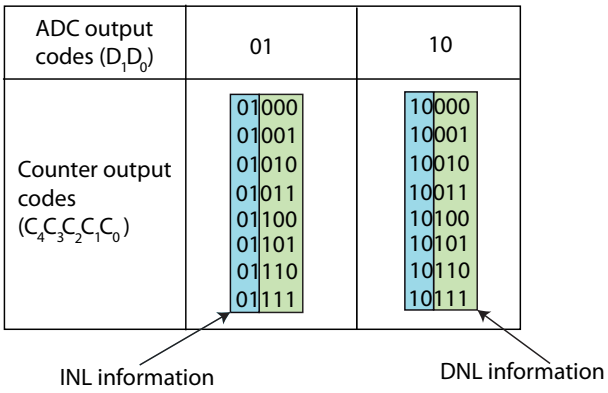


Figure 2.9: Counter based on-chip DNL and INL calculation [2, 3].

of the ADC output with the n most significant bits of the counter. As for DNL, it is evaluated by examining the $m - n$ LSBs of the counter. The output of the counter is initialised at the beginning of the test, and is compared to the ADC output as the analog input of the ADC increases. Before performing the test, an initial calibration is needed in order to adapt the input ramp slope to the counter. The linearity of the input ramp limits the precision of the measurement, and also no hardware results were presented.

Fully digital compatible BIST strategy

In [18], a test signal generator is presented that can target the measurement of a predefined code. The method is inspired from the standard servo-loop technique (Figure.2.10). The feedback loop is realised with dynamically matched low resolution and low accuracy DACs. The technique is compatible with digital testing environment.

The source generator is a reconfigurable low accuracy segmented current steering (SCS) DAC. The deterministic dynamic element matching (DDEM) technique is used to control the reconfiguration and improve the accuracy and linearity of the stimulus. The MSB array in the segmented current steering DAC is low resolution and thus in order not to limit the performance another low-resolution DAC is incorporated that generates extra linear steps at the output. Thus, the source generator consists of two blocks, the segmented DDEM DAC block and the n_d-bit dither DAC block.

Figure 2.10 shows the general architecture of the proposed methodology. Stimulus signals to the ADC under test are generated by adding together the outputs of the dither DAC and the segmented DDEM DAC. The 'digital control block' consists of a state machine and a memory and generates the digital codes for controlling the Segmented DAC and the dither DAC inputs. The 'test pattern generator' block generates the preset code k whose transition voltage is to be evaluated. The digital comparator compares the output of the ADC with the preset code k and sends the result to the control block. During measurement, under each DDEM configuration and dither input, the feedback loop will help find the desired input codes of the MSB and LSB arrays of

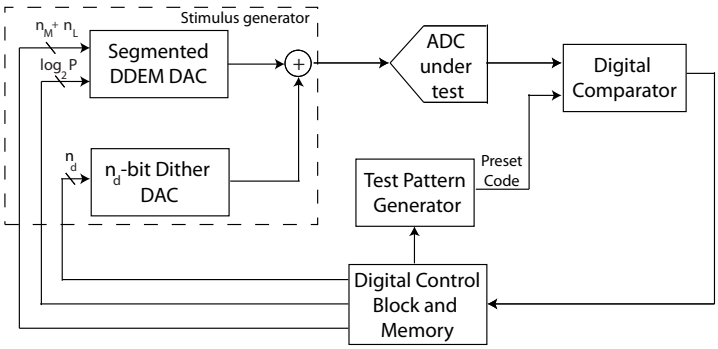


Figure 2.10: Fully digital compatible BIST strategy based on low resolution DDEM DACs.

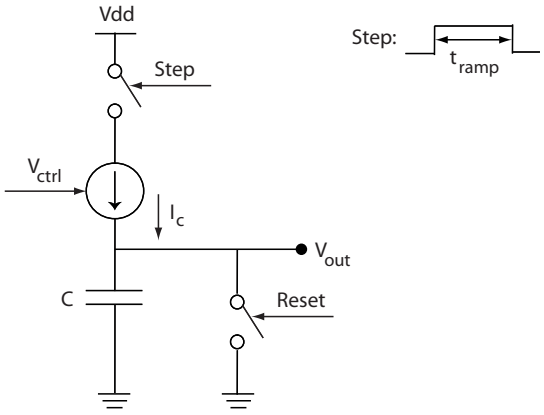


Figure 2.11: Basic principle of the ramp generation.

the segmented DDEM DAC. Thus, a stimulus sample that is closest to but less than the transition level T_k is generated. The digital input codes of the source generator are recorded in order to get the measurement of T_k. Binary search is used in order to find the adequate digital input codes of the source generator with fewer iteration cycles.

The testing performance is not sensitive to the mismatches and process variations, so that the analog BIST circuits can easily be reused without complex self-calibration. Simulation and experimental results show that the proposed circuitry can test the INL error of 12-bit ADCs to a ± 0.15 least significant bit (LSB) accuracy level using only 7-bit linear DACs.

The paper discusses mostly the design of the stimulus generator. Unfortunately, in the provided experimental results the authors did not put the entire DAC on one chip along with the ADC under test.

Integrated ramp generator for on-chip histogram

Integrating a ramp generator for on-chip histogram test has been investigated in [19], [20],[21],[22],[23] and [24]. The basic principle is to charge a capacitor with a constant

current source. Figure 2.11 shows the basic principle. The *Reset* signal initializes the output voltage and the *Step* signal controls the capacitor charging. The linearity of this circuit is limited by the output impedance of the current source, as this output impedance depends on the output voltage. For this purpose, [20] proposed a feedback amplifier configuration for charging the capacitor which makes the system less sensitive to the finite output impedance of the current source. This solution introduces an offset error that could be compensated with an auto-zero scheme.

The slope of the ramp is process dependant, thus, calibration schemes have been proposed to compensate for the process-dependant variability of the slope. The correction is performed by controlling the V_{ctrl} input with either a digital or an analog feedback system [19], [21].

The necessity to integrate a large capacitor that satisfies a long charging time is a limiting factor. Also, the best reported linearity is limited to 10 to 11 bits and it is challenging to design a system where current source range is adequate for testing the whole ADC range.

2.2.2 External Test Methodology or Design-For-Test

ADC histogram test using small triangular waves

In [25] and [26], Alegria et al. proposed a method where the ADC input range is stimulated by small-amplitude triangular waves superimposed to a progressively increased dc level. The test allows linearity constraints of stimulus generators to be relaxed and the experimental burden to be reduced.

The input range is scanned by progressively increasing the offset level step by step (Figure 2.12). The histogram procedure is adopted in order to reduce the sample number and the test duration in comparison to the standard static test. The number of samples to be acquired for the same uncertainty is much lower than the traditional static test.

In the proposed test, the constraint on the linearity of the triangular generator is relaxed by using a signal of amplitude much lower than the ADC full scale. The input range is stimulated entirely by acquiring the samples for the histogram in steps, with the same small-amplitude triangular wave, but with different offset levels.

The procedure of the proposed test is reported in Figure 2.13. First, the instruments are set. In each of the N_s steps, the ADC acquires M records of samples of the small wave with amplitude A. The sampling frequency and the small-wave frequency are selected according to the standard histogram procedure. The data acquisition is repeated N_s times for values of the offset C_j successively increasing. From the samples acquired in each step, a cumulative histogram CH_j $[k]$ is built. Calculating the transition levels from the cumulative histogram involves more calculations than a simple code hit counting (details can be found in [25]). The increase in the offset level from one step to another step should be precise but the authors do not discuss this aspect in the papers.

The elaboration of a complete BIST based on this methodology has not been proposed yet. In [27] and [28] generating full scale range triangular signals for BIST purposes has been inverstigated. The method by Alegria et al. could be combined with

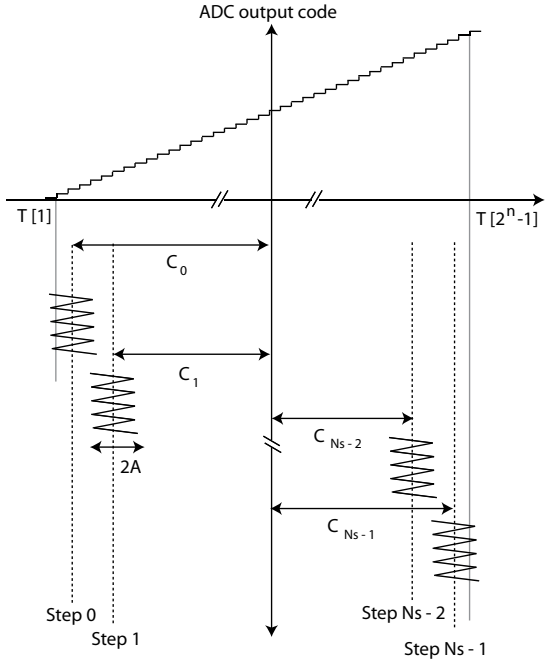


Figure 2.12: Signals applied to the ADC in the small-triangular waves based histogram.

the existing on-chip triangular signal generation techniques in order to make BIST of ADCs a more viable and less constraining challenge.

Stimulus error identification and removal

Conventional ADC linearity standards dictate that the test stimulus must be significantly more linear than the ADC under test (at least 2 bits more linear than the ADC under test). Satisfying this condition is more challenging for higher resolution ADCs. In [29] and [30] the authors introduce a method for ADC linearity test which permits using signal generators that may be significantly less linear than the device under test.

The method is referred to as SEIR (Stimulus Error Identification and Removal). Figure 2.14 shows the principle of the technique: two imprecise nonlinear test stimuli, $x_1(t)$ and $x_2(t)$, where $x_1(t) = x_2(t) + \alpha$ are applied to the ADC input to obtain two sets of ADC output data. $H_{k,1}$ for the input signal $x_1(t)$ and $H_{k,2}$ for the input signal $x_2(t)$. The SEIR algorithm then uses the redundant information from the two sets of data to accurately identify the nonlinearity errors in the stimuli. The algorithm then removes the stimulus error from the ADC output data, allowing the ADC nonlinearity to be accurately measured. For a high resolution ADC, the total computation time of the SEIR algorithm is significantly smaller than the data acquisition time and therefore does not contribute to testing time.

The approach was experimentally validated in production test on an industrial 16-

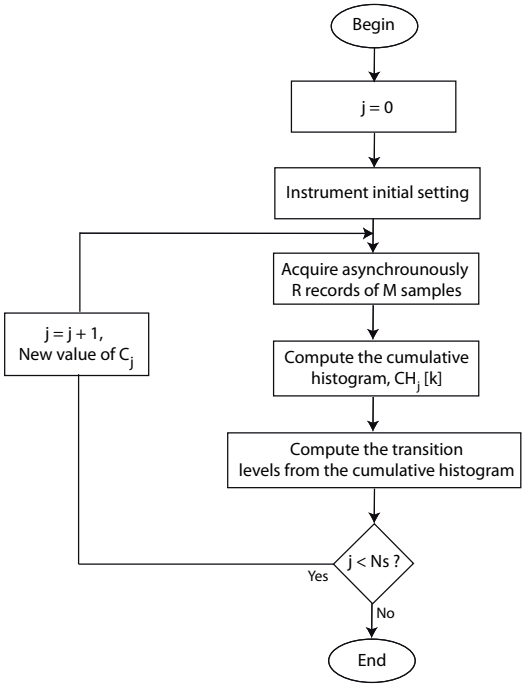


Figure 2.13: Test procedure of the small-triangular waves based histogram.

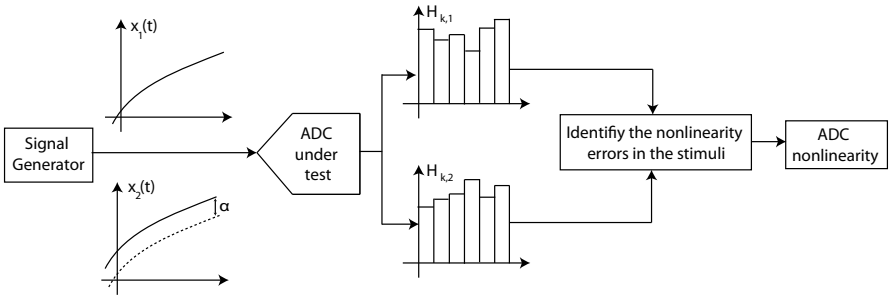


Figure 2.14: Principle of stimulus error identification and removal.

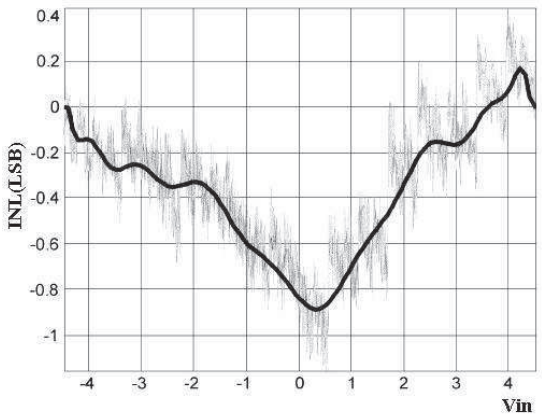


Figure 2.15: Comparison between the INL results of a 12-bit ADC using the technique in [4] (thick, smooth line) and results of the histogram technique.

bit successive approximation ADC by using 7-bit linear input signals. The use of such a technique is compromised by the required offset voltage stability which should be less than 0.1 LSB. It also requires too many computations for dedicated on-chip logic if considered to be inplemented in a BIST scheme.

Estimating static performances based on spectral test

In [4] the authors propose a technique for deriving the ADC Integral Non-Linearity from the outcome of the FFT test. The derived INL is a linear combination of Chebyshev polynomials, where the coefficients are the spurious harmonics of the output spectrum. The technique is convenient in the cases where the device under test has high resolution (16-20 bits) and a smoothed approximation of the INL is sufficient. The FFT test in this case is much faster than the standard static testing techniques for obtaining INL. Figure 2.15 shows the estimated INL for a 12-bit ADC using this technique. As it can be seen, the technique does not detect the maximum and minium INL usually derived when performing a standard static test.

In [31] an analysis was conducted to study the feasibility of an alternative static test flow involving exclusively spectral analysis. The methodology is based on a statistical approach to quantitatively evaluate the efficiency of detecting static errors from dynamic parameter measurements. The prediction of the test efficiency is based on a statistical analysis of the distribution of the measured dynamic parameters according to given tolerance limits for a wide population of ADCs.

It has been shown for example that complementing the classical FFT test by a second FFT test with non-conventional test conditions (increasing the stimulus amplitude, determination of DC offset and fundamental modulus) permits to enhance the efficiency of detecting the static-faulty ADCs. The method can not be generalized as its efficiency is strongly related to the ADC performances and to the considered ADC population.

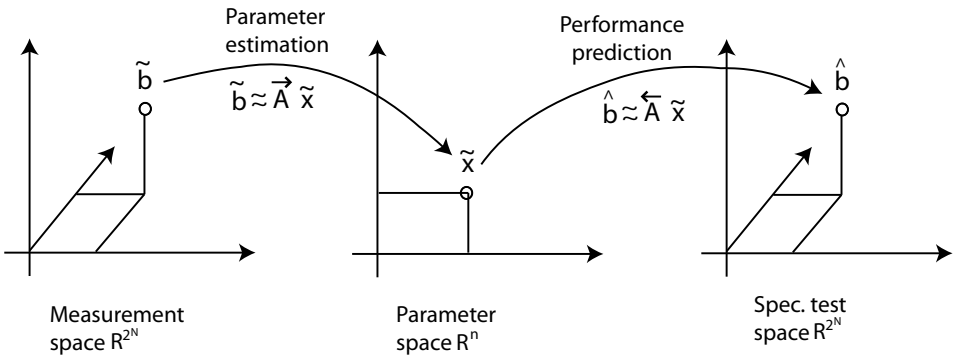


Figure 2.16: Linear model-based testing of an N-bit ADC [5].

Model based DNL and INL testing

The number of samples that needs to be acquired in the standard static test technique increases with the resolution of the ADC under test and the enviromental noise. Model based testing has been introduced as a mean to reduce the static test time of ADCs by reducing the number of samples that needs to be acquired and using a model to predict the performances of the ADC.

In [5], a model based technique is proposed and used in the objective of decreasing the test uncertainty without having to increase the number of acquired samples, as shown in Figure 2.16. First a 'noisy' measurement $\tilde{b}$ of the device characteristic is taken. From $\tilde{b}$ the least-squares estimate of the model parameter vector $\tilde{x}$ is obtained (since the solution is obtained in the least-square sense this provides the noise-reduction capability of this model-based test). Finally, $\tilde{x}$ is used to predict $\hat{b}$, an approximation of the unknown noise-free device characteristic. Thus, the uncertainty of the predicted all-codes INL characteristic $\hat{b}$ can be lower than the uncertainty of the measured characteristic $\tilde{b}$.

The work in [5] does not make any assumption on the type of the ADC. In [32], the authors propose a model based approach for testing pipeline ADCs (cf. Figure 3.1). The transfer characteristic of each stage j is modeled by the following equation:

$$V_{k-1} = g_{1j} * V_k + g_{dj} * V_{DACRef} + V_{offset} + g_{2j} * V_k^2 + g_{3j} * V_k^3 + ... \tag{2.12}$$

Where the g's represent the inverse of the gains. The non-linear error sources are represented by a Tailor series expression of the residue voltage. V_{DACRef} is the ideal DAC output voltage when an ideal DAC reference is used. V_{offset} models the sources of offset errors although in the proposed algorithm the authors use only one offset voltage for the entire pipeline. The parameters to be identified are the g's for each stage and the one V_{offset}. If M data points are used for the identification process, M error terms are formed based on an initial guess of the parameters compared to the actual measured ADC output code. Based on this, an optimization problem is set

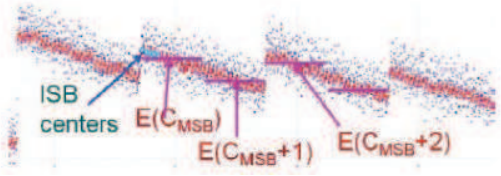


Figure 2.17: Segmented non-parametric model of ADC INL [6].

up in which the optimal parameters are solved that minimize the total summation of the squares of the expected values of the error terms. The problem then becomes a nonlinear statistical estimation problem and the Newton-Ralphson iteration method is used to solve the optimization problem. Once the parameters are identified, the model equations are used to compute the transition voltages of the ADC.

In [6] another model based ADC linearity test technique has been proposed. It is based on modeling the ADC's INL curve with a 'segmented non-parametric' model in which the INL curve is broken into many MSB segments according to the MSB value of the ADC output code, as shown in the example of Figure 2.17. The error with respect to the average value of each MSB segment is denoted by $E(CMSB)$ where $CMSB$ is the MSB code value in the MSB segment and it represents an error term in the segmented non-parametric model of the INL curve.

The authors make the assumption that the ADC is approximately linear. This assumption is verified in the beginning of the test procedure by checking the total power of the error signal after subtracting the expected linear code from the actual ADC output code, as shown in the flowchart of the algorithm in Figure 2.18.

When the ADC is approximately linear, each MSB segment can be modeled with the same inner structure, and some mechanisms are allowed to capture the slight gradual shape from segment to segment. Otherwise, further breaking down of the segments may be carried out. In this case the inner structure of an MSB segment can be modeled with the same segmented non-parametric model approach. In the example of Figure 2.17, the MSB segment is further broken into ISB segments with 'I' standing for intermediate. In doing so, more error terms are introduced in the overall non-parametric model. The authors state that 'in their experience' two levels of segmentation are sufficient but they do not provide a theoretical proof. If the segmentation is stopped at the ISB level, the variations within each ISB segment away from the ISB average values are captured by the LSB code width errors (WE in Equation. 2.13). Then the INL is given with:

$$INL(C) = E(C_{MSB}) + E_{C_{MSB}}(C_{ISB}) + WE(C_{LSB}) \tag{2.13}$$

With the model of Equation 2.13, it seems theoretically possible to compute the full code INL/DNL by identifying all the independent error terms in the model. The number of input output samples that need to be taken to identify the model depend on the resolution of the ADC, on how the segmentation is done, on how much ISB nonlinearity is allowed. For example, for 14, 16 and 18 bit ADCs it is in the range of 100 to 200 samples. However, the authors state that in an actual testing environment, several 100 times more data is needed to average out the noise.

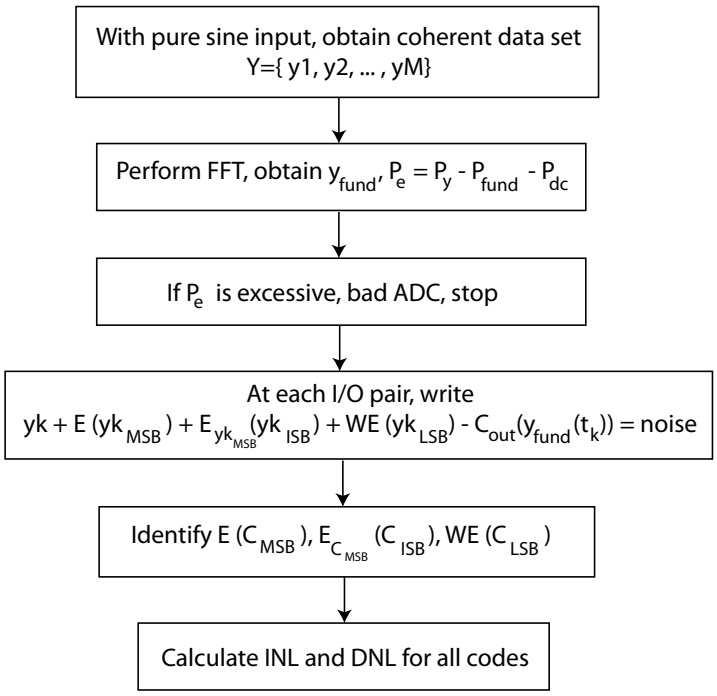


Figure 2.18: Flow chart of the proposed ADC test algorithm [6].

The authors show simulation and experimental results although they do not demonstrate theoretically the validity of their algorithm. Also, the algorithm can only be applied to ADCs having a segmented INL curve (SAR and pipeline ADCs) and requires complex calculations to obtain the estimated INL and DNL.

Reduced code testing

Some ADC architectures offer the possibility to apply what is known as reduced code testing. Static test time reduction is possible since we can rely on the measurements of a reduced set of judiciously selected code widths to extract the complete static characteristic of the ADC. The reported work has so far concerned SAR ADCs [33], [34] and pipeline ADCs [7], [35], [36]. Performing a successful reduced code testing scheme requires full understanding of the aspects of the ADC architecture.

Figure 2.19 shows the general architecture of a SAR ADC. The analog input is first compared to the mid-range (MSB of the N-bit register set to 1 and this forces the DAC output to be Vref/2). If the analog input is greater than Vdac, the comparator output is high and the MSB of the N-bit register remains at 1. Conversely, if the analog input is less than Vdac, the comparator output is low and the MSB of the register is set to 0. The SAR control logic then moves to the next bit in the register, sets it to 1 and compares to the next reference. This continues all the way down to the LSB. The ADC output code is then given by the values stored in the register.

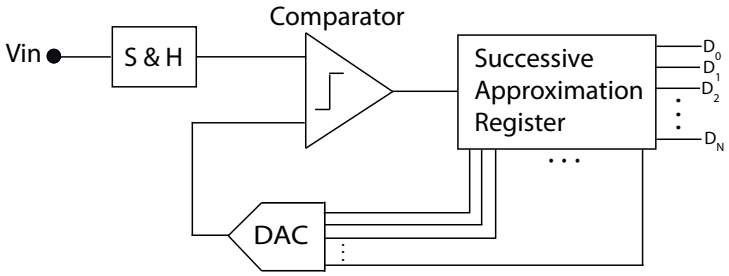


Figure 2.19: General architecture of a SAR ADC.

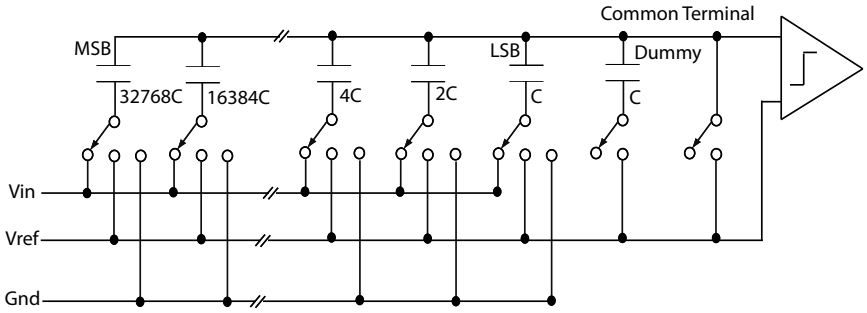


Figure 2.20: Capacitive binary weighted DAC.

The DAC shown in Figure 2.19 can be implemented as a unary or binary weighted DAC. In [33] a reduced code testing technique is presented that can be applied to SAR ADCs having a binary DAC implementation. Figure 2.20 shows a 16-bit binary weighted DAC implementation.

The reduced code testing technique presented in [33] is based on the fact that in a binary weighted DAC each capacitor is related to one bit and if the capacitor switch is turned on then the corresponding bit is set to 1. Every bit transition corresponds to the same capacitor being turned on or off in the DAC. The process variations affecting the capacitors will then affect the DNLs of the codes related the transitions of the same capacitor in the same way. Thus, it will be possible to measure few code widths and still be able to estimate the width of the rest of the codes. The authors claim that for an N-bit SAR ADC, only N code widths need to be measured.

SAR ADCs are relatively low resolution and slow. Pipeline ADCs are faster and have higher resolution. Reduced code testing is interesting in both cases because ADC test time gets higher when the ADC sampling frequency is low and also when the number of codes to be measured is high.

In our work we have developped a reduced code testing technique for pipeline ADCs. In the next chapter, the state-of-the art of reduced code testing of pipeline ADCs will be discussed in detail. We will show the limitations of the existing technique and we will describe the technique that we propose and that permits applying reduced code testing on pipeline ADCs in a very efficient way.

2.3 Discussion

As can be seen, many attempts have been suggested to solve the ADC static test problems. BIST to compute on-chip the static performances is a promising approach since it can eliminate the need to transfer a large volume of data off-chip to the automatic test equipment and it can also alleviate the problems related to noise and unstable electrical contacts. However, no satisfying solution has been reported to date. Model based techniques require complex computations and full fault coverage is not guaranteed. The small triangular waves and the Stimulus Error Identification and Removal approaches relax the requirements on the input stimulus linearity but they do not reduce the test time. Deducing the static performances from a spectral test can only be applied in some cases and can not be generalized.

The ADC static test problem gets exacerbated as the ADC resolutions are getting higher. Cutting down the number of codes that need to be measured for a certain ADC architecture will alleviate many of the test problems and will make some of the proposed test techniques in the literature a more viable option. ADCs are complex devices and thus trying to find a solution for ADC test without considering each ADC architecture thoroughly might not be fruitful.

In the following we will describe a very interesting property of pipeline ADCs and we will show how to exploit it in order to apply an efficient reduced code linearity test scheme.

Chapter 3

Reduced code linearity testing of pipeline ADCs

3.1 Overview of pipeline ADCs

A pipeline ADC consists of a cascade of stages as shown in Figure 3.1. Each stage consists of a sample-and-hold (S/H) circuit, a sub-ADC, a sub-DAC, a subtractor, and an amplifier. The input signal to each stage is first converted by the sub-ADC to a digital code which is the output of the stage. The result of the conversion is reconverted by the sub-DAC to an analog signal and subsequently subtracted from the input signal. The result of the subtraction is amplified so as to use the same reference voltage in all stages. The residue of the first stage is sampled by the second stage and so forth. The digital logic assembles the digital codes of the cascaded stages and provides the digital output of the ADC [37, 38, 39].

Each stage can generate 1-bit or multiple bits. The S/H, sub-DAC, substractor and amplifier are implemented as a switched-capacitor circuit which is commonly referred to as Multiplying Analog-to-Digital Converter (MDAC). Figure 3.2 shows the implementation of a 1-bit stage.

The MDAC block is controlled by a two phase clock to realize the function of sample and hold. During ϕ_1, the input signal is sampled into the capacitors C_f and C_s. During phase ϕ_2, a charge transfer takes place that results in a residue given by:

$$V_{out} = \frac{C_s + C_f}{C_f} * V_{in} + \frac{C_s}{C_f} * (1 - q_i * 2) * V_{ref} \tag{3.1}$$

where q_i is the digital output of the stage (in this case, $q_i = 0$ if $V_{in} < 0$; otherwise, $q_i = 1$).

The ratio $\frac{C_s+C_f}{C_f}$ defines the gain of the stage. In the general case, the residue of the stage is given by:

$$V_{out} = Gain_{stage} * (V_{in} - V_{dac}) \tag{3.2}$$

where V_{dac} is the analog representation of the digital output of the sub-DAC. In the case of the 1-bit stage, $V_{dac} = -V_{ref}/2$ if $V_{in} < 0$; otherwise, $V_{dac} = +V_{ref}/2$. The

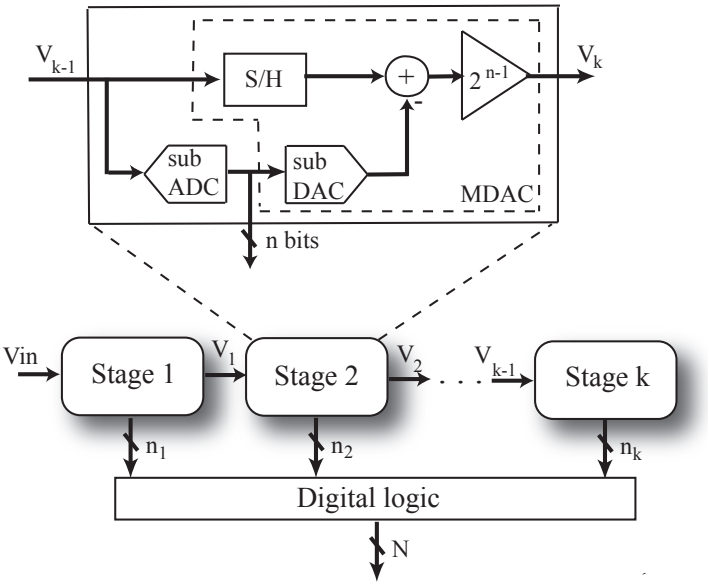


Figure 3.1: Architecture of a pipeline ADC.

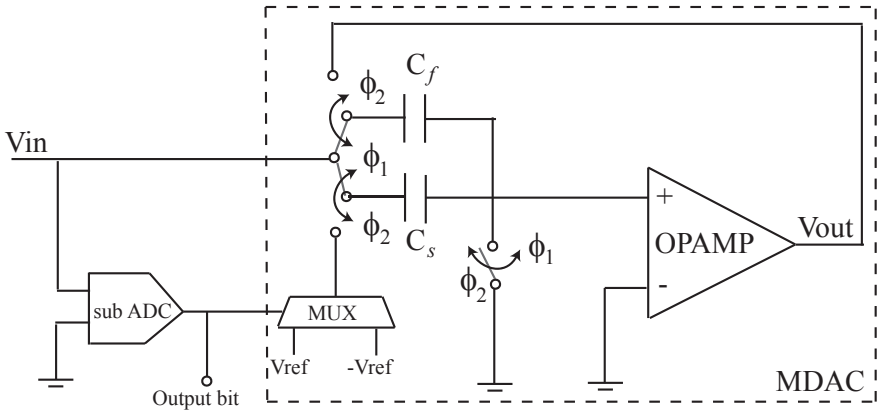


Figure 3.2: Implementation of the 1-bit stage.

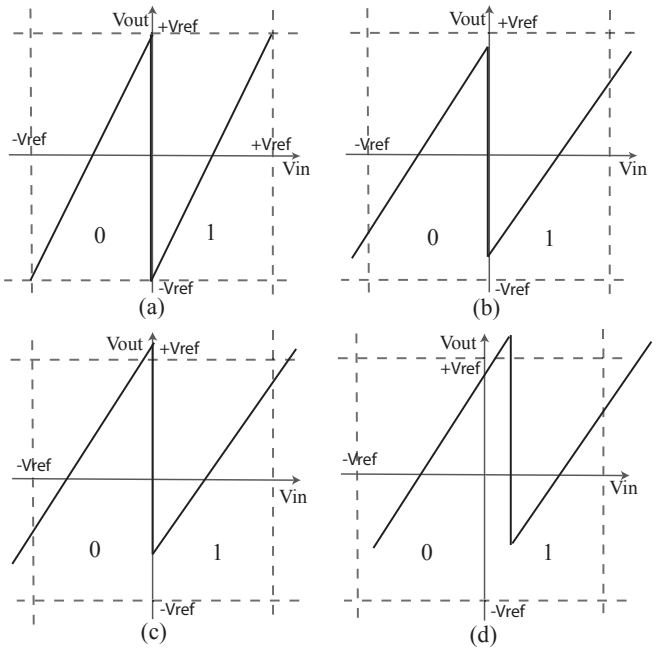


Figure 3.3: Residue of a 1-bit stage in the presence of non-idealities: (a) nominal; (b) the gain is lower than 2; (c) op-amp offset or charge injection; and (d) comparator offset.

residue of an ideal 1-bit stage is plotted in Figure 3.3(a). In Figure 3.3(b), 3.3(c), and 3.3(d) the residue is plotted considering different sources of error. A gain error due to finite op-amp gain or capacitor mismatch moves the residue peaks higher or lower, an offset in the op-amp or charge injection from the feedback switch shifts the residue vertically, and an offset in the comparator shifts the residue horizontally and vertically. The presence of such errors in the cascaded stages result in non-linearity errors in the ADC transfer characteristic [40, 41].

In the presence of these error sources the values of $Gain_{stage}$ and V_{dac} in Equation (3.2) will deviate from their ideal values [42, 43]. The comparator offset error will make the transition appear at a value of V_{in} that is shifted with respect to the ideal comparator thershold. The amplifier offset will shift the whole characteristic horizontally as follows:

$$V_{out} = Gain_{stage} * (V_{in} - V_{dac}) + V_{Offset_{Amp}} \tag{3.3}$$

The gain of the stage can be affected by the capacitor mismatch and the amplfier's finite gain and bandwidth. Assuming one pole transfer function, in the non-ideal case the gain of the stage is given by:

$$Gain_{stage} = \frac{C_s + C_f}{C_f} * (1 - e^{\frac{-t}{\tau}}) * (\frac{1}{1 + A * f}) \tag{3.4}$$

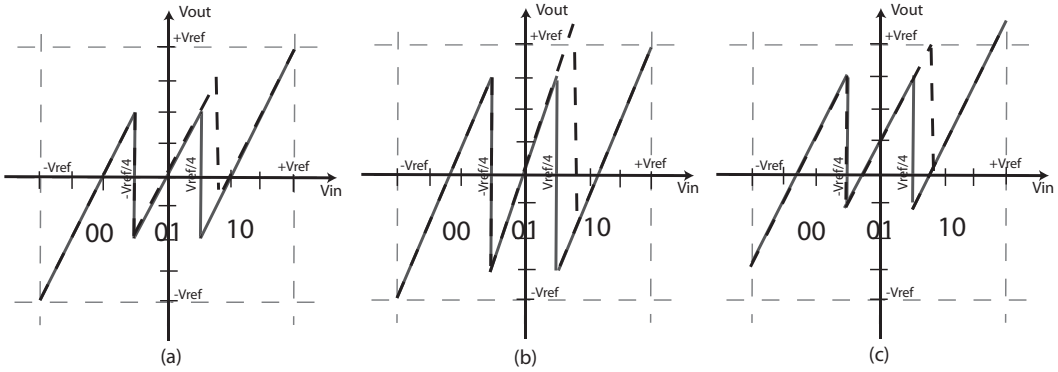


Figure 3.4: Residue of a 1.5-bit stage in the ideal case (continuous line in (a)) and in the presence of errors: continuous line in (b) assumes gain error, continuous line in (c) assumes op-amp offset, dashed line in (a) assumes comparator offset, dashed line in (b) assumes comparator offset and gain error, dashed line in (c) assumes comparator and op-amp offsets.

where A is the amplifier's finite gain, τ is related to the gain bandwidth product of the amplifer ($\tau = \frac{1}{BW}$), and f is the feedback factor ($f = \frac{C_f}{C_s+C_f}$).

We can observe from Figure 3.3 that the slightest error in the components of the stage may cause the residue to overrange. In order to moderate this effect digital correction is usually used in pipeline ADCs [44].

For the 1-bit stage for example, it consists of employing a second comparator in the sub-ADC. The thresholds of the two comparators are $-V_{ref}/4$ and $V_{ref}/4$ resulting in the residue shown with the continuous line in Figure 3.4(a). The residue shown with the dashed line in Figure 3.4(a) and the rest of the transfer functions in Figure 3.4(b) and (c) are drawn by taking into consideration the presence of different errors. Through the digital correction, larger comparator offset and gain errors can be tolerated before the residue over-ranges beyond $-V_{ref}$ or V_{ref}.

The amount of comparator offset or gain errors that can be tolerated depends on the combined effect of the different error sources. The 2-bit digital output can assume three of four possible levels, hence this stage is referred to as a 1.5-bit stage. The codes 00 or 10 ensure a certain 1 or a certain 0, while the code 01 shows that the bit determination is postponed and depends on the output of successive stages. The digital correction logic sums the digital outputs of the stages by making a 1-bit overlap between the digital outputs of the different stages.

In the case of a 2.5-bit stage the sub-ADC consists of 6 comparators [45]. The residue and digital output of a 2.5-bit stage are shown in Figure 3.5. Note that in the residues of a 1-bit (Figure 3.3(a)), 1.5-bit (Figure 3.4(a)) or 2.5-bit (Figure 3.5(a)) stages, everytime a comparator threshold is crossed, there is a transition in the digital output of the stage.

Figure 3.6 shows how the digital output code of a three-stage pipeline ADC is processed with and without digital correction for two different combinations of output codes. The weight which is associated to each stage depends on the position of the

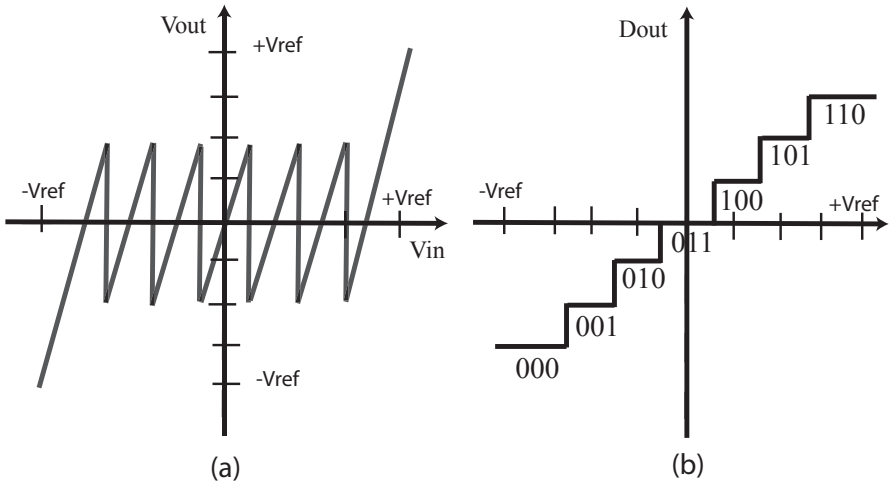


Figure 3.5: Residue (a) and digital output (b) of a 2.5-bit stage.

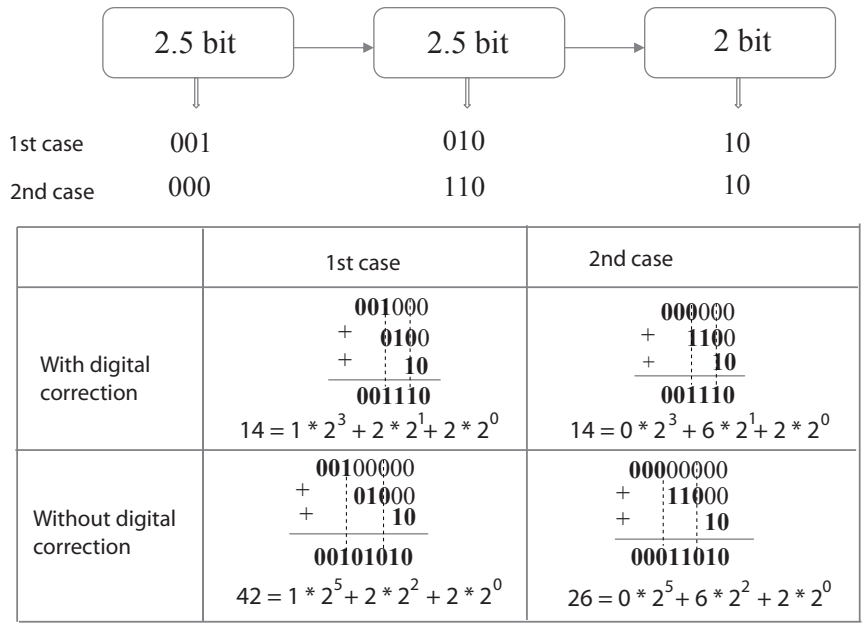


Figure 3.6: Effect of digital correction.

stage in the pipeline and on how the output codes are processed. For example, as shown in Figure 3.6, the weight of the first stage is 2^3 when digital correction is used and 2^5 when digital correction is not used. Note that when digital correction is used different combinations of stages digital outputs could result in the same ADC output code.

3.2 Principle of reduced code linearity testing of pipeline ADCs

3.2.1 Residues of pipeline ADC stages

Figure 3.7 plots together the residues of the first and the second stages of a 1.5-bit/stage pipeline ADC. The number placed above the peak of a transition indicates which of the two comparators is being exercised (e.g. its threshold is crossed) at this transition.

As can be seen, if we traverse the input dynamic range of the ADC, the two comparators of the first stage are exercised once. The first, second, and third segment of the first stage residue traverse the output ranges $[-V_{ref}, V_{ref}/2]$, $[-V_{ref}/2, V_{ref}/2]$, and $[-V_{ref}/2, V_{ref}]$, respectively. Thus, for each segment of the first stage residue, each of the two comparators of the second stage is exercised once. Therefore, if we traverse the input dynamic range of the ADC, the two comparators of the second stage are exercised three times each in total. Following a similar argument, if we traverse the input dynamic range of the ADC, the two comparators of the third stage are exercised seven times each. This can be seen in Figure 3.8 where the residue of the third stage is included too.

Figure 3.9 plots together the residues of the first two stages of a 2.5-bit/stage pipeline ADC. A 2.5-bit/stage comprises six comparators and provides a 3-bit output. As can be seen, the six comparators in the first stage are exercised once if we traverse the input dynamic range of the ADC. In contrast, in the second stage, the first and sixth comparators are exercised just once while the rest of the comparators are exercised seven times each.

3.2.2 The principle

The bottom line of the above discussion is that a comparator in a second or later pipeline stage is exercised several times. This implies that in the ADC output there will be transitions that are due to the same comparator, as shown in Figure 3.10.

The ADC output transitions that involve the same comparator are all affected in the same way in the presence of errors due to process variations. Thus, to extract the complete transfer characteristic, it suffices to consider only a representative set of output transitions which covers all comparators in all stages.

Each comparator in each stage needs to be represented only once in this set. For the selected set of ADC output transitions, we measure the widths of the codes around them. Thereafter,we can readily use these measurements to infer the widths of the codes around the transitions that were not selected. In the example of Figure 3.10,

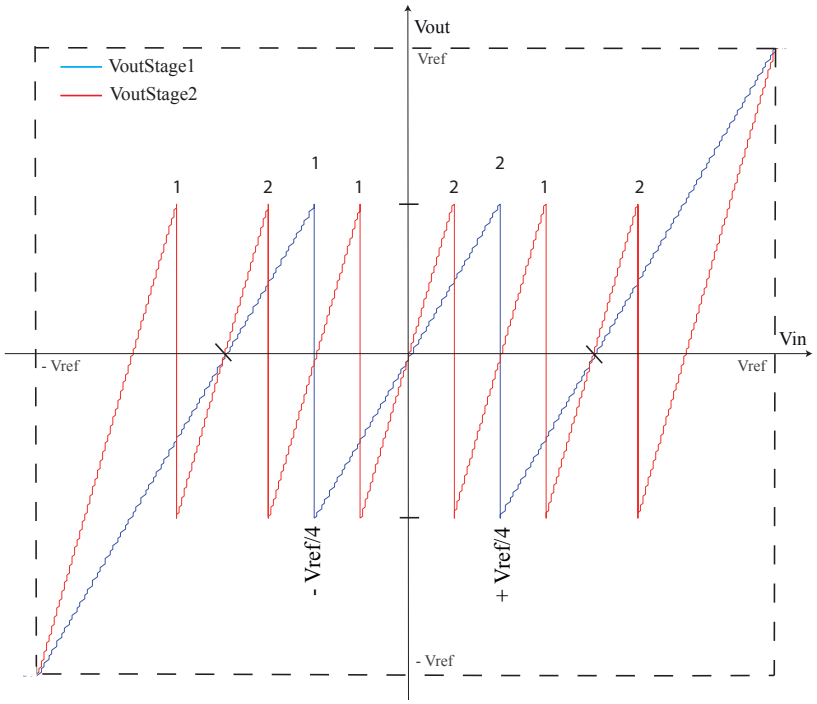


Figure 3.7: Residue of the first and second stages of a 1.5-bit/stage pipeline ADC.

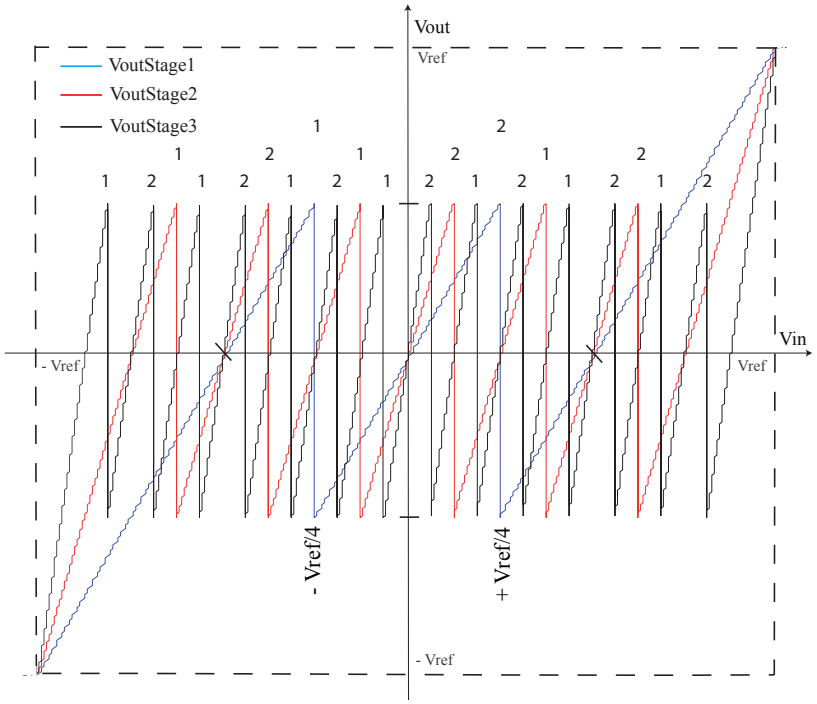


Figure 3.8: Residue of the first three stages of a 1.5-bit/stage pipeline ADC.

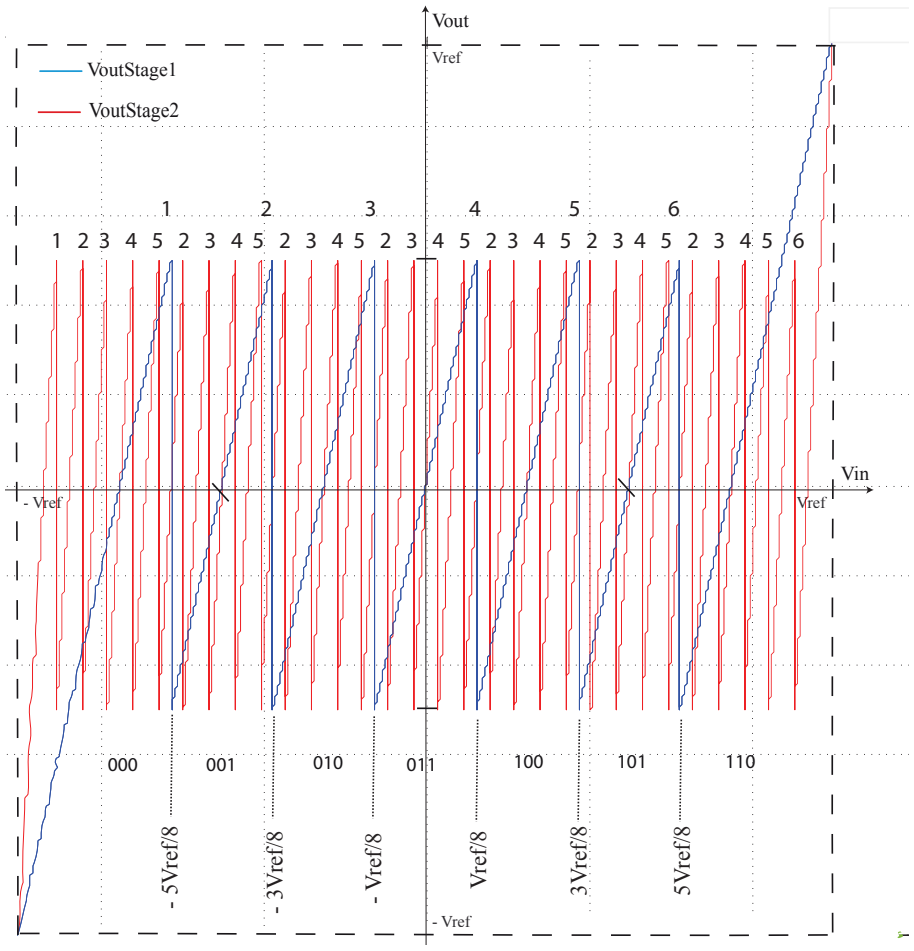


Figure 3.9: Residue of the first two stages of a 2.5-bit/stage pipeline ADC.

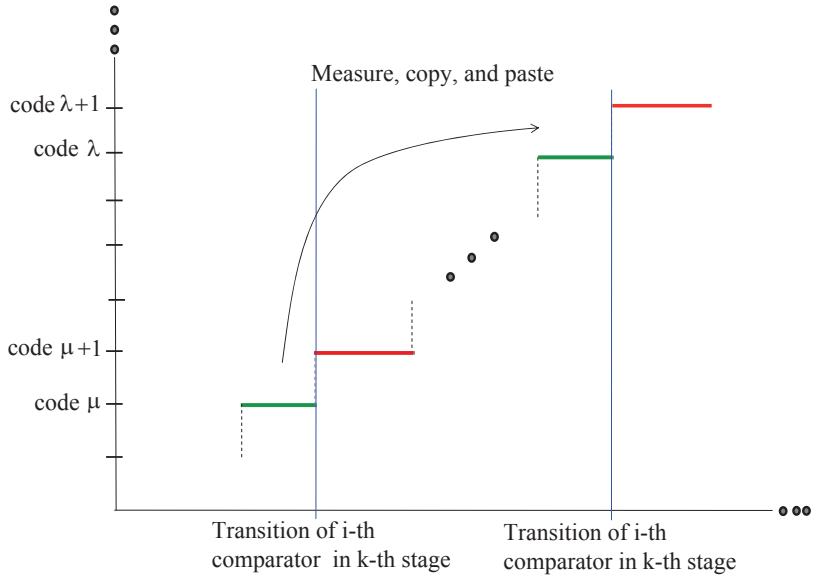


Figure 3.10: Principle of reduced code testing of pipeline ADCs.

the widths of the codes λ and $\lambda + 1$ are equal to the widths of the codes μ and $\mu + 1$, respectively, since the two ADC output transitions are due to the same comparator.

Thus, we need to measure the width of either λ or μ and the width of either $\lambda + 1$ or $\mu + 1$. In this way we fill in the histogram by relying on a reduced number of code width measurements which translates in linearity test time reduction.

Of course, a code shares two adjacent transitions that involve two different comparators in different stages and the question that arises is which comparator will be considered for mapping the ADC output transition to a comparator transition. Since a stage in the pipeline dominates all subsequent stages in terms of the error produced in the transfer characteristic, we choose the comparator that belongs to the stage that is closer to the beginning of the pipeline. In practice, to obtain good accuracy, it may be necessary to consider more than two codes around the transitions that involve comparators which belong to the first stages of the pipeline. The number of the codes to be considered depends also on the amount of errors that are present in the ADC which is reflected on the maximum and minimum DNL and INL. The larger the DNL and INL errors are and the more towards the front of the pipeline the stage is, the larger this number is recommended to be.

In order to make the reduced code testing technique successful, we need to meet two objectives. First, we need to ensure that an ADC output transition is mapped to the correct comparator. This holds for all ADC output transitions, i.e. those that are selected and those that are not selected and whose surrounding codes will be inferred later. Second, for a comparator in a given stage we should avoid selecting an output transition that involves in addition to this comparator a comparator in one of the previous stages. The reason is that the error of the previous stage will overshadow

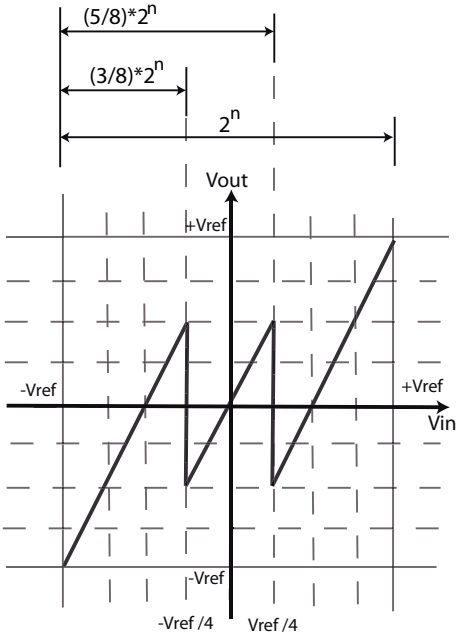


Figure 3.11: Locating the transitions of the comparators of a first 1.5-bit stage with respect to the complete transfer characteristic of the ADC [7].

the error of the target stage. If the above two objectives are not met, then the accuracy of the technique degrades, resulting in an erroneous characterization of the static performances of the ADC.

3.3 State-of-the-art of reduced code testing of pipeline ADCs

3.3.1 The existing approach

The possibility to apply reduced code testing on pipeline ADCs was first introduced in [7]. In this paper the authors propose to map the ADC output transitions to the corresponding comparator transitions by simply locating the transitions of the comparators on the ADC transfer characteristic. For example, as shown in Figure 3.11, the ADC output codes corresponding to the transitions of the comparators of a first 1.5-bit stage will be situated at codes:

$(((\frac{3}{8}) * 2^n)^{th}, ((\frac{3}{8}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{5}{8}) * 2^n)^{th}, ((\frac{5}{8}) * 2^n + 1)^{th})$ (comparator 2)

of the ADC transfer characteristic.

Similarly, the codes corresponding to the transitions of the comparators of a second 1.5-bit stage will be situated at codes:

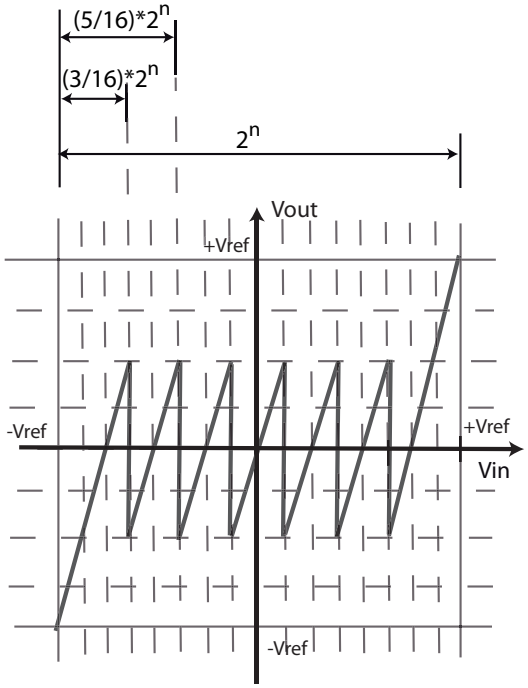


Figure 3.12: Locating the transitions of the comparators of a first 2.5-bit stage with respect to the complete transfer characteristic of the ADC [7].

$(((\frac{3}{16}) * 2^n)^{th}, ((\frac{3}{16}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{5}{16}) * 2^n)^{th}, ((\frac{5}{16}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{7}{16}) * 2^n)^{th}, ((\frac{7}{16}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{9}{16}) * 2^n)^{th}, ((\frac{9}{16}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{11}{16}) * 2^n)^{th}, ((\frac{11}{16}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{13}{16}) * 2^n)^{th}, ((\frac{13}{16}) * 2^n + 1)^{th})$ (comparator 2)

of the ADC transfer characteristic.

In the case of a first 2.5-bit stage, the comparators transitions will be situated at codes: (see Figure 3.12)

$(((\frac{3}{16}) * 2^n)^{th}, ((\frac{3}{16}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{5}{16}) * 2^n)^{th}, ((\frac{5}{16}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{7}{16}) * 2^n)^{th}, ((\frac{7}{16}) * 2^n + 1)^{th})$ (comparator 3)
$(((\frac{9}{16}) * 2^n)^{th}, ((\frac{9}{16}) * 2^n + 1)^{th})$ (comparator 4)
$(((\frac{11}{16}) * 2^n)^{th}, ((\frac{11}{16}) * 2^n + 1)^{th})$ (comparator 5)
$(((\frac{13}{16}) * 2^n)^{th}, ((\frac{13}{16}) * 2^n + 1)^{th})$ (comparator 6)

of the ADC transfer characteristic.

Figure 3.9 shows that in a second 2.5-bit pipeline stage the first and the sixth comparators are exercised only once while the 4 other comparators are exercised seven times each. For a second 2.5-bit pipeline stage the first and the sixth comparators transitions will be situated at codes:

$(((\frac{3}{64}) * 2^n)^{th}, ((\frac{3}{64}) * 2^n + 1)^{th})$ (comparator 1)
$(((\frac{61}{64}) * 2^n)^{th}, ((\frac{61}{64}) * 2^n + 1)^{th})$ (comparator 6)

of the ADC transfer characteristic, while the comparators from 2 to 5 occupy the intermediate positions:

$(((\frac{5}{64}) * 2^n)^{th}, ((\frac{5}{64}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{7}{64}) * 2^n)^{th}, ((\frac{7}{64}) * 2^n + 1)^{th})$ (comparator 3)
$(((\frac{9}{64}) * 2^n)^{th}, ((\frac{9}{64}) * 2^n + 1)^{th})$ (comparator 4)
$(((\frac{11}{64}) * 2^n)^{th}, ((\frac{11}{64}) * 2^n + 1)^{th})$ (comparator 5)
$(((\frac{13}{64}) * 2^n)^{th}, ((\frac{13}{64}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{15}{64}) * 2^n)^{th}, ((\frac{15}{64}) * 2^n + 1)^{th})$ (comparator 3)
$(((\frac{17}{64}) * 2^n)^{th}, ((\frac{17}{64}) * 2^n + 1)^{th})$ (comparator 4)
..
..
..
$(((\frac{53}{64}) * 2^n)^{th}, ((\frac{53}{64}) * 2^n + 1)^{th})$ (comparator 2)
$(((\frac{55}{64}) * 2^n)^{th}, ((\frac{55}{64}) * 2^n + 1)^{th})$ (comparator 3)
$(((\frac{57}{64}) * 2^n)^{th}, ((\frac{57}{64}) * 2^n + 1)^{th})$ (comparator 4)
$(((\frac{59}{64}) * 2^n)^{th}, ((\frac{59}{64}) * 2^n + 1)^{th})$ (comparator 5)

of the ADC transfer characteristic.

This mapping method is only valid if there are no comparator offsets at all. The above mentionned locations are ideal and they will be shifted from their ideal positions due to the presence of comparator offset. The same authors have proposed another mapping technique that is not affected by the presence of comparator offset in [35] and [36].

To locate the transitions of a the comparators of a certain stage it is suggested to force the LSB of the digital outputs of this stage to zero. Looking at the transfer characteristic of the ADC this will result in apparent discontinuities located at the transitions of the comparators of the stage under observation.

As an example, Figure 3.13 shows the transfer characteristic of an ideal 4-bit, 1.5-bit/stage pipeline ADC together with the resultant transfer characteristic when the LSB of the digital output of the second stage is forced to zero. As expected from Figure 3.7, we observe six jumps: each of the two comparators of the second stage results in three of them.

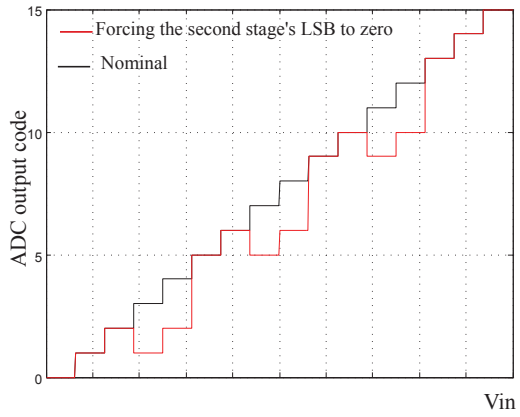


Figure 3.13: The transfer function of a 1.5-bit/stage ADC when the LSB of the digital codes of the second stage is forced to zero.

Figure 3.14 shows the transfer characteristic of a 12-bit, 2.5-bit/stage pipeline ADC together with the resultant transfer characteristic when the LSB of the digital output of the first stage is forced to zero (red curve) and the resultant transfer characteristic when the LSB of the digital outputs of the second stage is forced to zero (black curve). As expected from Figure 3.9, we observe 6 jumps on the red curve and they correspond to the six comparators of the first stage. As for the second stage, we observe 30 jumps on the black curve and they correspond to the repeated occurrences of some of the comparators of this stage (referring to Figure 3.9, there is a total of 30 transitions in the second stage).

To map the located ADC output transitions to the corresponding comparators it is suggested in [36] to consider the expected distribution of comparators transitions. As an example, Figure 3.15 and Figure 3.16 show the ideal locations and the tolerated variation ranges of the comparators transitions of the first three stages in a 1.5-bit/stage pipeline ADC and of the first two stages in a 2.5-bit/stage pipeline ADC, respectively. This approach is valid when the comparators of the stage are exercised in the expected order. As we will see later, particularly for a 2.5-bit/stage pipeline ADCs, this assumption is rarely met.

3.3.2 Limitations of the existing approach

To select the representative comparator transitions it is suggested in [36] to consider again the expected distribution of comparator transitions, as shown in Figure 3.15 and 3.16. In reality, the comparators transitions will be shifted horizontally away from their ideal locations due to comparator offset, as shown in Figure 3.4. If we neglect all other sources of error, the arrows in Figure 3.15 and Figure 3.16 indicate the maximum variation range of each comparator transition that will be tolerated by digital correction. If we take into consideration all sources of error, the tolerated variation range for each comparator transition can be narrower or larger compared to the equal variation ranges

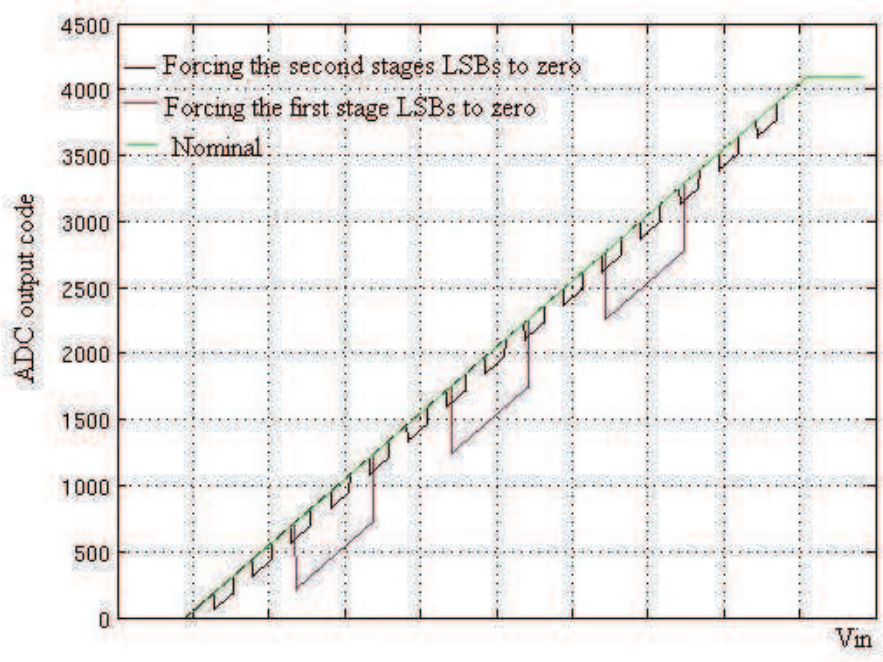
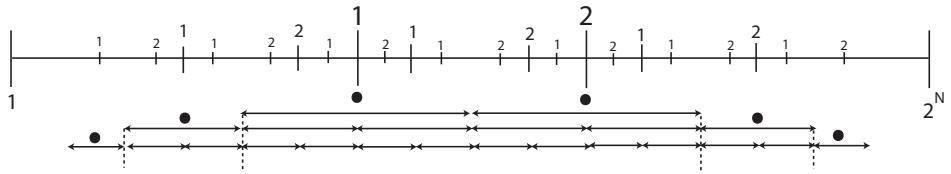
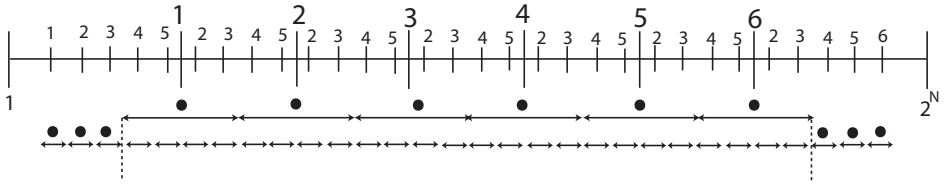


Figure 3.14: The transfer characteristic of a 2.5-bit/stage ADC when the LSB of the digital codes of the first and second stages are forced to zero.



The selected transitions are shown with ●
The number x with {large, intermediate, small} font corresponds to the ideal transition edge for comparator x in the {first, second, third} stages, respectively

Figure 3.15: Distribution and selection of transitions for the first three stages of a 1.5-bit/stage pipeline ADC.



The selected transitions are shown with ●
The number x with {large, small} font corresponds to the ideal transition edge for comparator x in the {first, second} stages, respectively

Figure 3.16: Distribution and selection of transitions for the first two stages of a 2.5-bit/stage pipeline ADC.

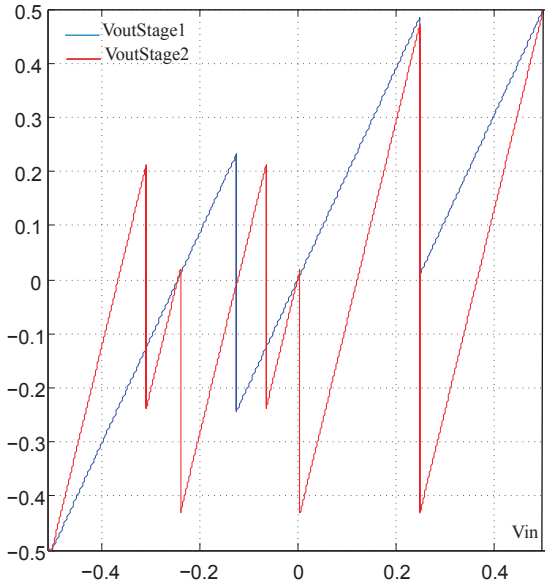


Figure 3.17: Residues of the first and second stages of 1.5-bit/stage pipeline ADC in the presence of errors.

shown in Figure 3.15 and Figure 3.16.

When considering only comparator offset and when this offset can be tolerated by digital correction, the optimal selection of comparators transitions is shown with dots in Figure 3.15 and Figure 3.16. This selection principle is optimal under these assumptions since each selected comparator transition lies in an area that is empty of previous stages comparator transitions. Thereby, a transition is always mapped to the correct comparator and it is impossible that it overlaps with a transition corresponding to a comparator of a previous stage.

The above approach to decide which comparator transitions to consider is problematic when the assumptions do not hold. In the following we will illustrate what can go wrong in practice with a few examples. The simulations are carried out using a behavioral model of a pipeline ADC provided by STMicroelectronics.

We consider first a 12-bit 1.5-bit/stage pipeline ADC consisting of 11 cascaded 1.5-bit stages and a last 1-bit flash ADC. We suppose that the first and second stages have both a gain smaller than 2. In addition, the second comparator of the first stage has a positive offset and the second comparator of the second stage has a negative offset. The simulation result is shown in Figure 3.17. As can be seen, the transition of the second comparator in the first stage overlaps with the last transition in the second stage. In this scenario, the selection principle will result in an erroneous estimation of the non-linearity of the ADC.

Next we consider a 12-bit 2.5-bit/stage pipeline ADC consisting of 5 cascaded 2.5-bit stages and a last 2-bit flash ADC. In this case the problem is exacerbated since, as shown in Figure 3.16, the tolerated variation ranges of the comparators transitions are smaller compared to those of a 1.5-bit/stage pipeline ADC.

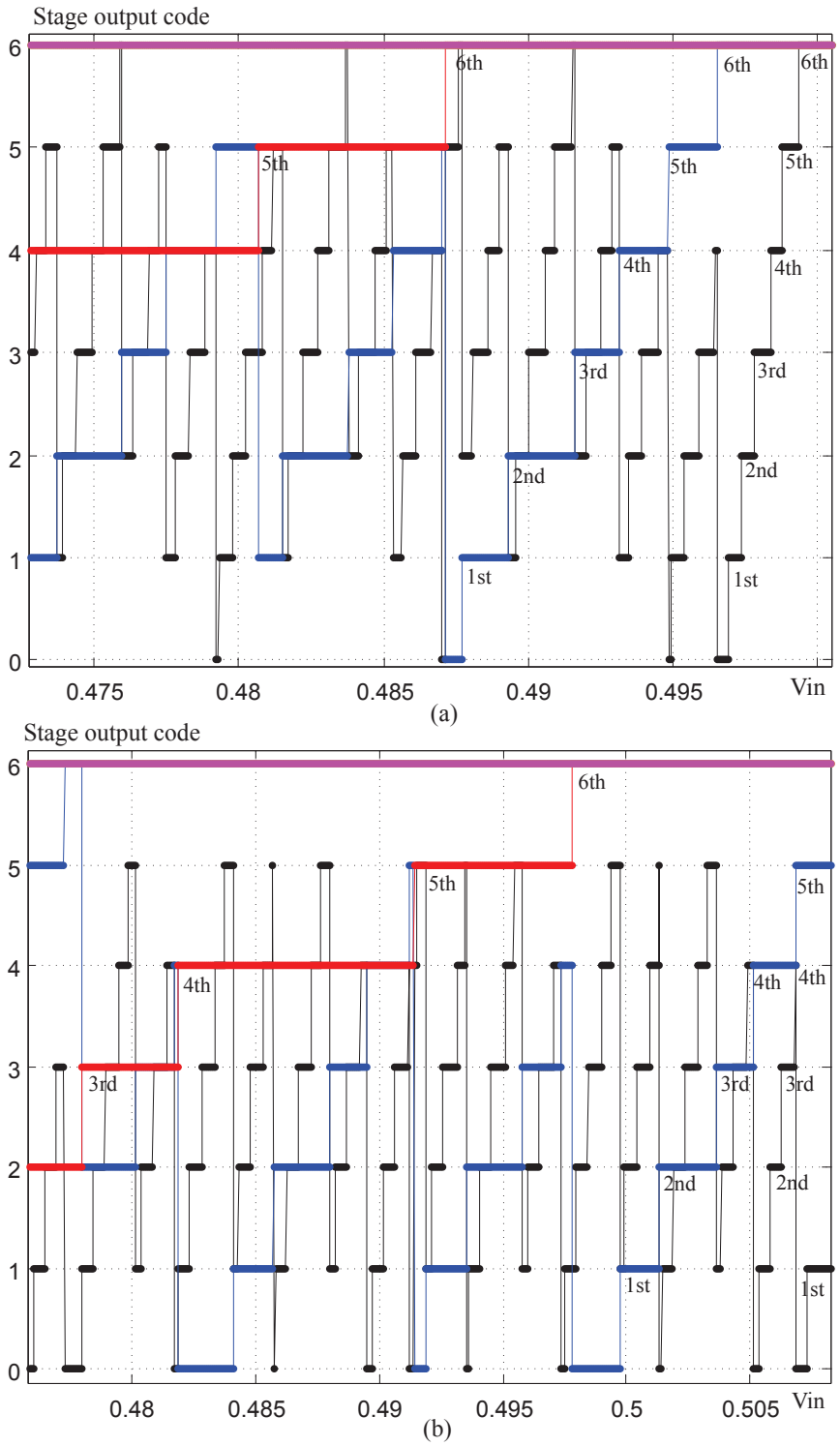


Figure 3.18: Digital output of the first five stages of a 12-bit pipeline ADC towards the end of the dynamic range: (a) considering only comparator offset and (b) considering all error sources (Magenta: first and second stages; red: third stage; blue: fourth stage; black: fifth stage)

Let us consider Figure 3.18(a) which shows the digital outputs (in decimal from '1' to '6') of the first five stages of a 2.5-bit/stage pipeline ADC in the presence of comparator offset only. The plot is a zoom of the transfer functions of the five stages towards the end of the dynamic range of the ADC (the plot would be unreadable if we had considered the whole dynamic range since the simulated ADC has a high resolution).

The output of the first and second stages are fixed to '6' in the zoomed interval since we are looking at the end of the dynamic range. In contrast, we can observe the last transitions of the other stages. The numbers placed next to the transitions indicate the comparator of the stage that is exercised. In this case, the selection principle of Figure 3.16 will work fine since the last three transitions that will be selected in each stage are mapped to the correct comparators.

Now, if in addition to comparator offset we consider gain error and op-amp offset, we obtain Figure 3.18(b). As can be seen, the three last transitions of the fourth stage now correspond to its third, fourth, and fifth comparator, respectively. Furthermore, the first of the last three transitions of the fifth stage corresponds to its fourth comparator and the last transition corresponds to its first comparator. Therefore, the 'blindly' selected transitions for the fourth and fifth stages will be mapped to the wrong comparators which will result in an erroneous estimation of the non-linearity of the ADC.

In the presence of different error sources the comparators do not always get exercised in the expected order. Forcing the LSB of the output of the stage to zero and then relying on an expected order of comparators transitions to perform the mapping will result in an erroneous linearity estimation. The order in which the comparators transitions will happen depend on the combination of error sources that are present in the pipeline stages and thus in order to insure full fault coverage we can not rely on any expected order.

3.4 Natural and forced transitions

The second of the last three transitions of the fifth stage in Figure 3.18 (b) correspond to a change of the digital output of the stage from '4' to '0'. This transition does not correspond to crossing the threshold of one of the comparators in the fifth stage. To explain this, let us consider the ideal residues of the first two stages of a 1.5-bit/stage pipeline ADC, as shown in Figure 3.19(a). The figure also shows the output of the sub-DAC of the first two stages which changes whenever the digital output of the stage changes.

Looking at the residue of the second stage, we can observe the six transitions corresponding to the two comparators. The first comparator is exercised three times (e.g. transitions $00 \rightarrow 01$) and the second comparator is exercised also three times (e.g. transitions $01 \rightarrow 10$). Even though there are only six compartors being exercised in the stage, we observe that the digital output of the stage, in addition to the transitions $00 \rightarrow 01$ and $01 \rightarrow 10$, contains two other transitions (e.g. transitions $10 \rightarrow 00$). These two transitions of the digital output do not correspond to a transition in the residue of the second stage, but they happen under the influence of transitions in the residue

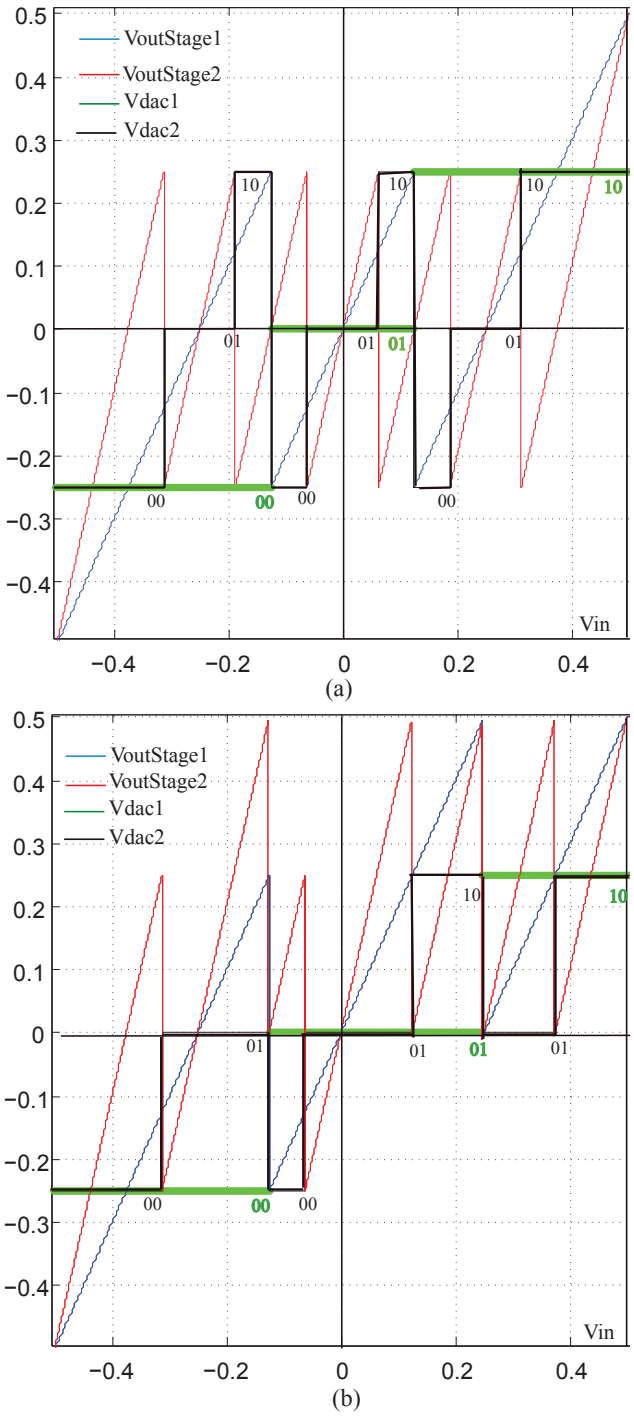


Figure 3.19: Residues of the first two stages of a 1.5-bit/stage pipeline ADC plotted together with the output of the sub-DAC at the time of code transitions: (a) nominal; (b) in the presence of comparator offset.

of the first stage (see Figure 3.19(a)). The residue of the first stage becomes suddenly lower than the threshold of the second comparator in the second stage and thus the digital output of the second stage transitions from 10 to 00.

We refer to these transitions as *forced* transitions because the digital output of the second stage changes, yet this change is not due to crossing the threshold of one of the second stage comparators. Conversely, when the digital output of the stage changes due to one of its comparators being exercised, the transition is referred to as *natural* transition.

Figure 3.19(b) shows how the residues are transformed due to the presence of comparator offset. As can be seen, two of the six transitions of the digital output of the second stage are *forced* transitions.

3.5 The enhanced reduced code linearity test technique

As mentioned earlier, in order for the technique to succeed we need to meet two objectives. First, we need to ensure that an ADC output transition is mapped to the correct comparator. Second, in order to select an ADC output transition that is representative of a comparator in a given stage we should avoid selecting an ADC output transition that involves in addition to this comparator a comparator in one of the previous stages.

To meet these two objectives and to avoid pitfalls such as those demonstrated in the previous section, we propose to monitor the digital output of each of the stages before it undergoes digital correction, as shown in Figure 7.13. The rationale is that when a comparator threshold is crossed it necessarily produces a transition in the digital output of the stage to which it belongs. A transition in the digital output of a stage provides complete information about which comparator has been exercised. Furthermore, a transition in the digital output of a stage can be mapped to the resulting ADC output transition by simply processing the outputs of the different stages as is done by the digital logic block of the ADC.

Since we rely on the transitions which occur in the digital output of each stage, we need to list and label all the possible transitions that may happen. If we consider an ascending ramp, a *natural* transition corresponds to a rise by one step of the digital output of the stage. If the stage digital output decreases or jumps by more than one step, then the transition is a $forced$ transition. As explained above, a forced transition at the output of a stage is due to a natural transition that has occurred in a previous stage.

Table 3.1 lists all possible transitions that may happen in the output of a 2.5-bit stage during the operation of the ADC assuming an ascending ramp. The digital output of the stage at a certain time is denoted by X, whereas the digital output at the next clock cycle is denoted by Y. There are six possible natural transitions corresponding to the six comparators denoted by $C1$ to $C6$. Fxy indicates a forced transition from code x to code y. The shaded boxes indicate that the digital output has not changed. Overall, digital monitoring should indicate which of the 42 different scenarios in Table 3.1 has occurred for each of the stages.

Figure 3.21 shows an example on how to use the information collected by digital

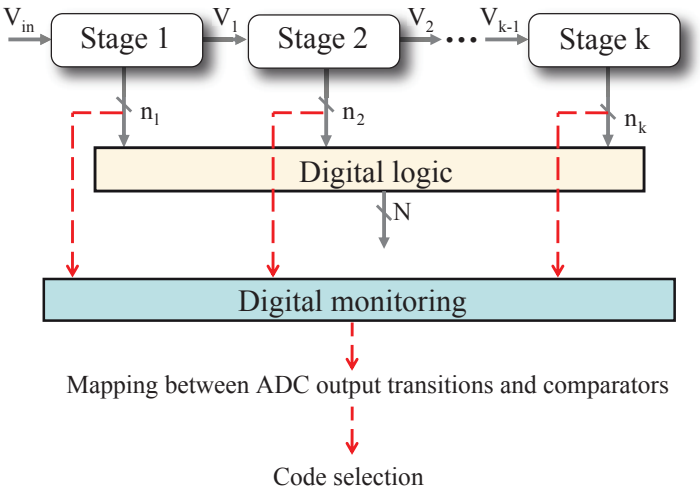


Figure 3.20: Digital monitoring aiming at correct mapping between ADC output transitions and comparators.

	T1	T2	T3
Stage1 transition type:	C2	0	0
Stage 2 transition type:	F23	C3	0
Stage 3 transition type:	C4	F52	C4

C2 of stage 1 | C3 of stage 2 | C4 of stage 3

Figure 3.21: Example on how to use the information collected by digital monitoring.

Table 3.1: Digital output transition table of a 2.5-bit stage.

Y \ X	000	001	010	011	100	101	110
000		F10	F20	F30	F40	F50	F60
001	C1		F21	F31	F41	F51	F61
010	F02	C2		F32	F42	F52	F62
011	F03	F13	C3		F43	F53	F63
100	F04	F14	F24	C4		F54	F64
101	F05	F15	F25	F35	C5		F65
110	F06	F16	F26	F36	F46	C6	

monitoring. Suppose that we have captured the type of the transitions for the first three stages at three different times T1, T2 and T3. We recall that by selecting an ADC output transition for a comparator in a certain stage we should ensure that this ADC output transition does not involve other comparators in any of the previous stages. At T1 the resulting ADC output transition will be mapped to the second comparator in the first stage, at T2 it will be mapped to the third comparator in the second stage, and at T3 it will be mapped to the fourth comparator in the third stage.

In summary, the reduced code testing technique consists of the following steps:

1. Find the mapping between the ADC output transitions and the comparators that are being exercised.

2. Select the representative set of ADC output transitions. Each comparator in each stage should be represented once in this set. For each comparator in stages 2 to N, where N is the total number of stages, there is a number of transitions that we can select from. Considering a comparator in the k^{th} stage, we should avoid selecting an ADC output transition that involves in addition to this comparator a comparator in one of the previous stages.

3. Measure the widths of the codes around the selected ADC output transitions. The default is to measure the two codes on the left and on the right of an ADC output transition. However, in practice, it is recommended to take the measurement of more than two codes around the ADC output transitions that are mapped to the comparators of the first stages. The larger the DNL and INL errors are and the closer the stage which contains the comparator is to the beginning of the pipeline, the larger this number is recommended to be.

4. Infer the widths of the codes around the ADC output transitions that were not selected in step 2. A code has two adjacent ADC output transitions that are mapped to two different comparators in different stages. To fill in the value of the width of the code, we select the transition that is mapped to the comparator that belongs to the stage that is closer to the beginning of the pipeline.

5. Calculate DNL and INL from the obtained code widths.

Selecting the representative set of ADC output transitions (step 2) and inferring the widths of the codes around the ADC output transitions that were not selected (step 4) both depend on the mapping obtained in step 1. Step 1 is very important since the accuracy of the obtained results depend on the accuracy of the mapping obtained in this step. In this chapter we have shown the limitations of the mapping method proposed in [36] and we have proposed a new mapping method that is based on monitoring the digital output of the individual stages, where the rationale is that when a comparator threshold is crossed it necessarily produces a transition in the digital output of the stage.

The first two steps of the technique are carried out as follows. We first find the mapping between the ADC output transitions and the comparators that are exercised. For this purpose, we apply a ramp and we monitor the transitions that are happening at the digital output of each stage. Once the information on the transitions has been collected, we select a representative set of stage transitions and, thereafter, we map this set to the corresponding ADC output transitions by simply processing the stage outputs as is done by the digital logic of the ADC. Note that there is no linearity or precision constraints on the input signal that is intended to monitor the transitions that are happening at the digital output of the stages. The goal is just to make the comparators of the stages get exercised and the signal just needs to be monotonically increasing.

3.6 Simulation results

We consider the 12-bit 2.5-bit/stage pipeline ADC whose stages transitions were discussed in Section 3.3.2. Figure 3.22(a) and 3.23(a) show the DNL and INL using the standard histogram technique.

The codes that were identified as having the maximum and minimum code widths had code widths of 31 hits and 2 hits, respectively.

These codes correspond to the transitions of the sixth comparator of the fourth stage, which does not appear in the last three transitions of this stage (Figure 3.18(b)). Thus, by using the mapping technique in [36] we would have estimated a maximum code width with 20 hits and a minimum code width with 11 hits. Thus, we would have underestimated the true non-linearity of the ADC.

The estimated DNL and INL using the reduced code linearity test technique that we propose are shown in Figure 3.22(b) and 3.23(b). As can be observed, the codes at which the maximum linearity error occurs have been correctly captured and the maximum and minimum DNL and INL have been correctly estimated.

Notice that the estimated DNL profile is more regular than the one obtained with the histogram, i.e. the DNL is estimated to be the same for many codes. This is due to the fact that we have selected just one ADC output transition representative of each comparator and we have replicated the measured code widths around this ADC output transition into the code widths around the ADC output transitions that are mapped to the same comparator. This results in minor differences in the estimated DNL and INL.

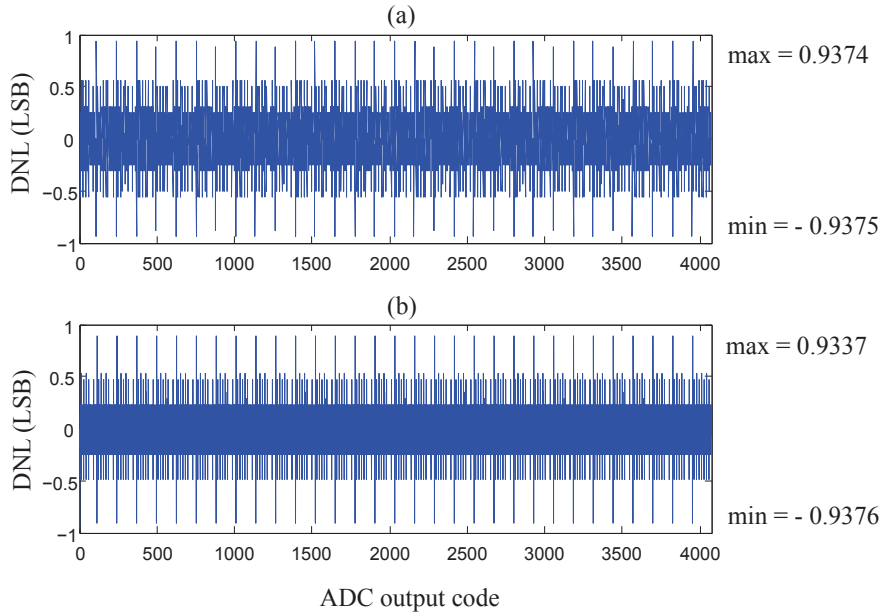


Figure 3.22: DNL of a 12-bit 2.5-bit/stage pipeline ADC: (a) standard histogram technique, and (b) the proposed technique.

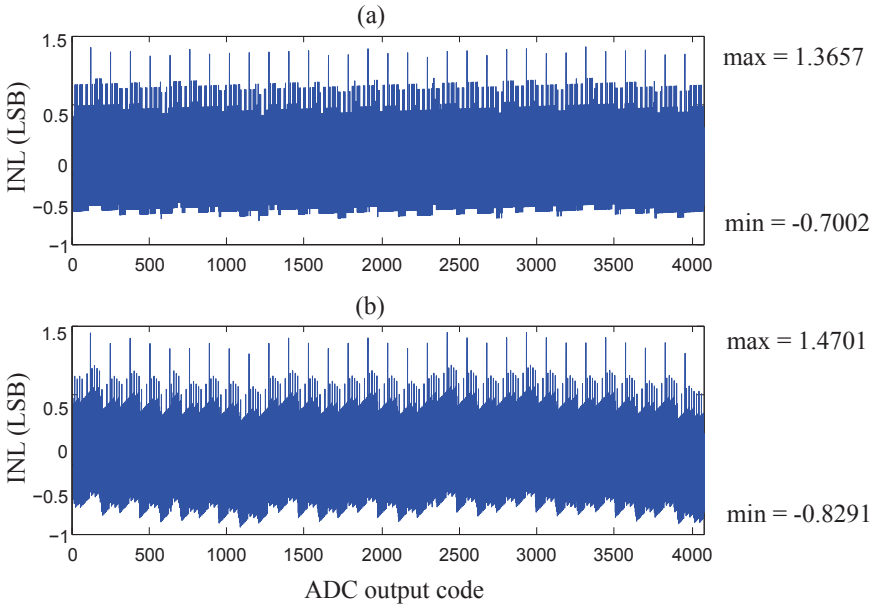


Figure 3.23: INL of a 12-bit 2.5-bit/stage pipeline ADC: (a) standard histogram technique, and (b) the proposed technique.

This difference can be made smaller at the expense of increasing the number of samples per code for the histogram technique that has been used to obtain the measurements of the selected code widths.

Chapter 4

Noise cancelling method

The effectiveness of the enhanced reduced code linearity testing technique was demonstrated in the previous Chapter using a behavioral model. In this Chapter we will show the inaccuracies encountered when experimenting in a real noisy environment. Next, we will show another important property of pipeline ADCs and we will present a method that exploits it and which permits circumventing the inaccuracies of estimation that may occur due to the presence of noise.

4.1 Noisy transitions

As it was emphasized in the previous chapter, the crucial step of the technique is to perform a correct mapping between the ADC output transitions and the corresponding comparators. Once a mapping is obtained, it is possible to group the codes that have the same width.

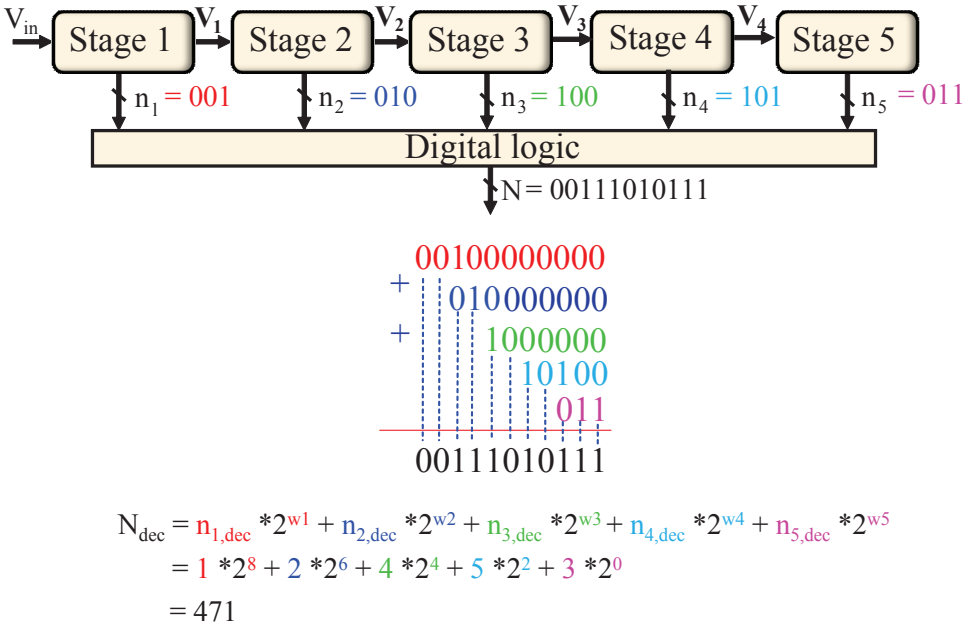
As described earlier, we rely on monitoring the transitions that are happening at the digital output of the individual stages. From this, it is possible to calculate the corresponding ADC output codes to the right and to the left of a comparator transition.

In Figure 4.1 we show a 5-stage pipeline ADC where each stage delivers a 3-bit output. The ADC output code is calculated by making a 1-bit overlap between the outputs of the stages. The digital output of the ADC can be expressed in decimal as a function of the outputs of the individual stages as:

$$N_{dec} = n_{1,dec} * 2^{w1} + n_{2,dec} * 2^{w2} + n_{3,dec} * 2^{w3} + n_{4,dec} * 2^{w4} + n_{5,dec} * 2^{w5} \qquad (4.1)$$

In the following we will refer to w_k as being the weight of stage k and to the coefficient $n_{k,dec}$ as being the contribution of stage k. In Figure 4.1 the calculation of the ADC output code corresponding to the given example is shown. The weight of stage k is equal to the sum of the number of bits of the following stages, minus the number of the overlapping bits from this stage onwards.

Figure 4.2 shows a snapshot of the transitions of the third, fourth, and fifth stages of an experimental 11-bit ADC which consists of four 2.5-bit stages and a last 3-bit stage (a different scale is used for each stage, the experimental setup will be explained



$$N_{dec} = n_{1,dec} *2^{w1} + n_{2,dec} *2^{w2} + n_{3,dec} *2^{w3} + n_{4,dec} *2^{w4} + n_{5,dec} *2^{w5}$$
$$= 1 *2^{8} + 2 *2^{6} + 4 *2^{4} + 5 *2^{2} + 3 *2^{0}$$
$$= 471$$

Figure 4.1: ADC output as a function of the contribution and weight of each stage.

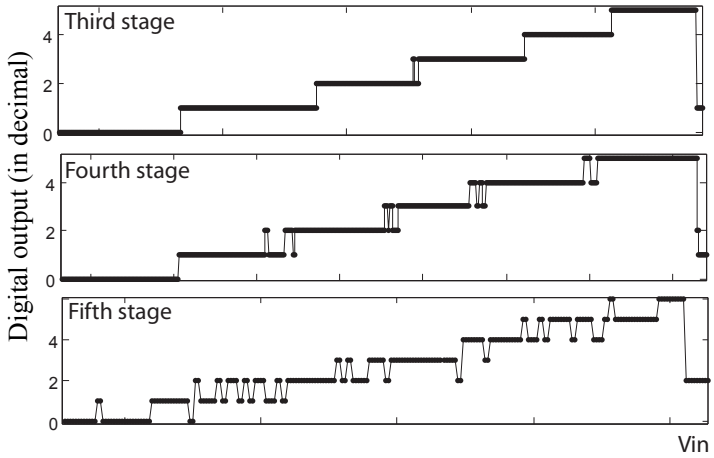


Figure 4.2: The effect of noise on the transitions (a different scale is used for each stage).

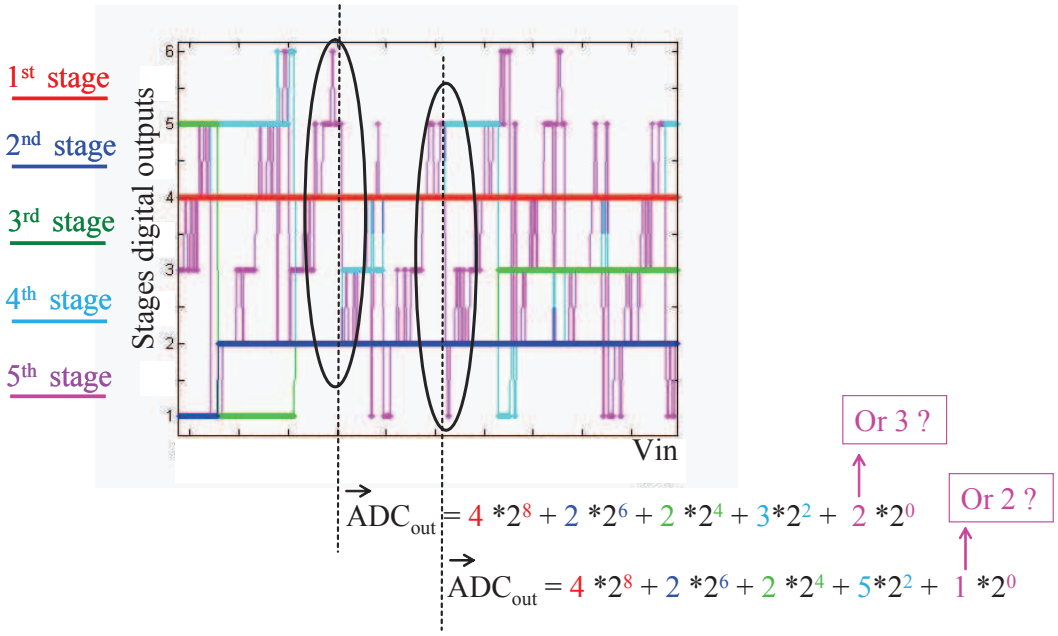


Figure 4.3: The effect of noise on the transitions (the outputs of all the stages are superimposed).

in Chapter 5). As can be seen, the transitions become noisier for stages that are towards the end of the pipeline (corresponding to the LSBs). The transitions of the first and second stages are very 'clean' and are not shown in Figure 4.2. In Figure 4.3 we superpose the digital outputs of the stages of the experimental ADC. This figure zooms in a part of the whole dynamic range. We will use Figure 4.3 to explain what can go wrong while performing the mapping in the presence of noise.

Each stage output (in decimal) is shown in a different color. We can see that some of the transitions are very noisy and this leads to innacuracies in the mapping. We show this with two examples in Figure 4.3. In the first example (first ADC_{out} from the left) the intention was to calculate the ADC output code to the right of the transition of the third comparator in the fourth stage (light blue curve, transition from '2' to '3'). It can be seen that:

- the output of the first stage to the right of this transition is '4',
- the output of the second stage to the right of this transition is '2',
- the output of the third stage to the right of this transition is '2',
- the output of the fourth stage to the right of this transition is '3'.

As for the fifth stage, if we read the output value that is immediately to the right of the transition we find a value of '2'. But visually, it could have been a value of '3'

too. Due to the presence of noise, the digital output of the fifth stage to the right of the transition under consideration is toggling between '2' and '3'.

In the second example (second ADC_{out} from the left) the intention was to calculate the ADC output code to the right of the transition of the fifth comparator in the fourth stage (light blue curve, transition from '4' to '5'). It can be seen that:

- the output of the first stage to the right of this transition is '4',
- the output of the second stage to the right of this transition is '2',
- the output of the third stage to the right of this transition is '2',
- the output of the fourth stage to the right of this transition is '5'.

As for the fifth stage, if we read the output value that is immediately to the right of the transition we find a value of '1'. But visually it seems that the value of '1' is a glitch and that the value that should be taken is '2'.

Thus, in the presence of noise, considering a stage output immediately to the left or to the right of a transition may result in an erroneous mapping between a comparator transition and ADC output transition. This will further result in an incorrect grouping of the ADC output codes. If the mapping is not correct, the estimation of DNL will be inaccurate, as we can attribute to a code a DNL value that is very different from its actual DNL value. Furthermore, successively summing up erroneous values of DNL may result in a significant error in INL estimation.

In the following, we describe another important property of pipeline ADCs and we show how to exploit it to cancel out the effect of noise while performing a reduced code testing scheme.

4.2 Root codes

For the explanations in this section we use a noise-free behavioural model of a 10-bit ADC that comprises four 2.5-bit stages and a last 2-bit stage. Let us consider the third comparator in the second stage of this ADC.

Figure 4.4 superimposes the output of the ADC (right y-axis in decimal) on the output of the second stage (left y-axis in binary) as we traverse the whole dynamic range. From this plot we can identify the output codes of the ADC that are associated with each of the transitions of the third comparator of the second stage (the output of the second stage transitions from 010 to 011 when its third comparator is exercised, see Table 3.1).

On the right y-axis of Figure 4.4 we show the ADC output codes on the right of the third comparator transition. The first ADC output code 112 corresponds to the case when the output of the first stage is 000, the second ADC output code 240 corresponds to the case when the output of the first stage is 001, the third ADC output code corresponds to the case when the output of the first stage is 010, and so forth. (Figure 4.7 helps visualize the output of the first stage with respect to the output of the second stage).

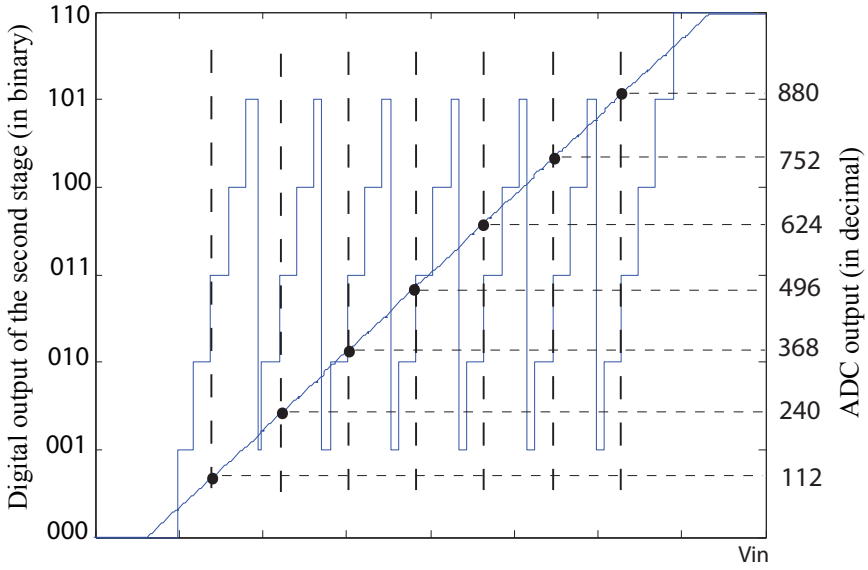


Figure 4.4: Transitions in the second stage and corresponding ADC output codes.

For this ADC the weight of the first stage is equal to 2^7. We can write the ADC output codes shown in Figure 4.4 under the form:

$$
\begin{aligned}
112 &= 112 + 0 \cdot 2^7,\\
240 &= 112 + 1 \cdot 2^7,\\
368 &= 112 + 2 \cdot 2^7,\\
496 &= 112 + 3 \cdot 2^7,\\
624 &= 112 + 4 \cdot 2^7,\\
752 &= 112 + 5 \cdot 2^7,\\
880 &= 112 + 6 \cdot 2^7.
\end{aligned}
$$

Note that the coefficient that is multiplied by the weight of the first stage in the above equations corresponds to the value of the output of the first stage in the zone where the ADC output code is taken. All ADC output codes shown in Figure 4.4 can be obtained from code 112 by adding a term that is obtained by multiplying the value of the output of the first stage by the weight of the first stage.

We refer to the output code 112 as the *right root code* of the third comparator in the second stage. Similarly, by looking at the ADC output codes on the left of the transition of the third comparator in the second stage, we can define the *left root code* of the third comparator in the second stage.

Let us now consider the third comparator in the third stage of the same ADC. Figure 4.5 superimposes the output of the ADC (right y-axis in decimal) on the output of the third stage (left y-axis in binary) as we traverse the whole dynamic range. From this plot we can identify the ADC output codes that are associated with each of the transitions of the third comparator of the third stage (the output of the third stage

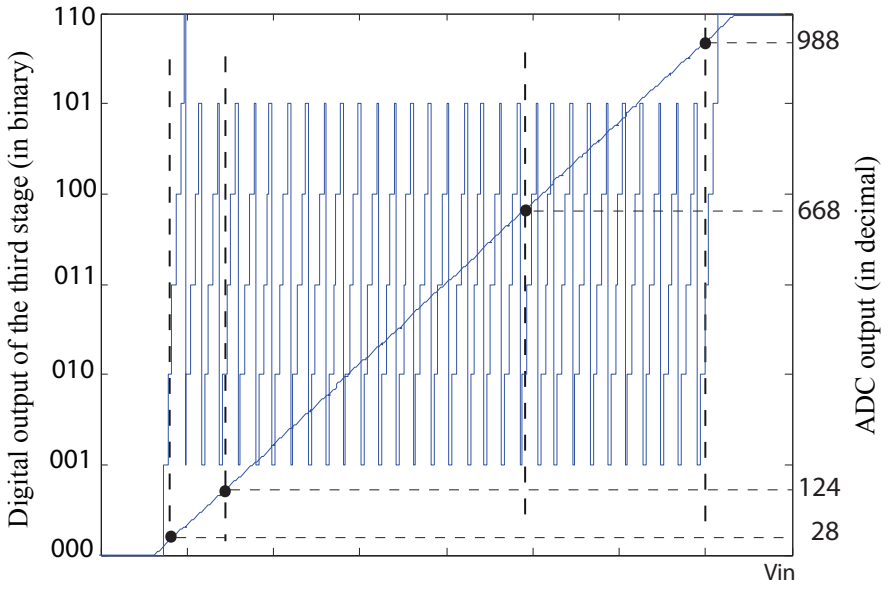


Figure 4.5: Transitions in the third stage and corresponding ADC output codes.

transitions from 010 to 011 when its third comparator is exercised).

The first ADC output code 28 corresponds to the case where the outputs of both the first and the second stages are 000. The rest of these codes can be expressed on the basis of the code 28 noticing that for this ADC the weight of the first stage is equal to 2^7 and the weight of the second stage is equal to 2^5. For example, the ADC output code 124 corresponds to the case when the outputs of the first and second stages are 000 and 011, respectively, that is:

$$124 = 28 + 0 \cdot 2^7 + 3 \cdot 2^5.$$

The ADC output code 668 corresponds to the case when the output of both the first and second stages is 100, that is:

$$668 = 28 + 4 \cdot 2^7 + 4 \cdot 2^5.$$

The ADC output code 988 corresponds to the case when the output of both the first and second stages is 110, that is:

$$988 = 28 + 6 \cdot 2^7 + 6 \cdot 2^5$$

and so forth for the other ADC output codes related to the transitions of the third comparator of the third stage (not shown in Figure 4.5).

We refer to the ADC output code 28 as the right root code of the third comparator in the third stage. Similarly, by looking at the ADC output codes on the left of the

transition of the third comparator in the third stage, we can define the *left root code* of the third comparator in the third stage.

To generalize, let us divide the ADC stages into two groups. The first group contains stages 1 to $k-1$ and the second group contains stages k to N, where N is the total number of stages. Let us also define a function $f(x, comp_k^i, w)$ where:

- x refers to the number of the stage in the pipeline,
- $comp_k^i$ refers to i^{th} comparator of the k^{th} stage,
- w refers to the right side R or to the left side L of the transition of comparator $comp_k^i$.

We define $f(x, comp_k^i, w)$ as follows:

> "Given a transition of the i^{th} comparator of the k^{th} stage, $f(x, comp_k^i, w)$ is the digital output of the x^{th} stage on the w side of the transition."

For example, $f(1, comp_2^3, R)$ refers to the digital output of the first stage on the right of a transition of the third comparator of the second stage. If we refer to the examples given on Figure 4.3, ADC_{out} is expressed as following for the first example from the left:

$$\begin{aligned} ADC_{out} = {} & f(1, comp_4^3, R) * 2^8 + f(2, comp_4^3, R) * 2^6 + f(3, comp_4^3, R) * 2^4 \\ & + f(4, comp_4^3, R) * 2^2 + f(5, comp_4^3, R) * 2^0 \end{aligned}$$

and as following for the second example from the left:

$$\begin{aligned} ADC_{out} = {} & f(1, comp_4^5, R) * 2^8 + f(2, comp_4^5, R) * 2^6 + f(3, comp_4^5, R) * 2^4 \\ & + f(4, comp_4^5, R) * 2^2 + f(5, comp_4^5, R) * 2^0 \end{aligned}$$

where, for the first example from the left: $f(1, comp_4^3, R) = 4$, $f(2, comp_4^3, R) = 2$, $f(3, comp_4^3, R) = 2$, $f(4, comp_4^3, R) = 3$ and $f(5, comp_4^3, R) = 2(or 3)$. And for the second example from the left: $f(1, comp_4^5, R) = 4$, $f(2, comp_4^5, R) = 2$, $f(3, comp_4^5, R) = 2$, $f(4, comp_4^5, R) = 5$ and $f(5, comp_4^5, R) = 1(or 2)$.

Every time the i^{th} comparator is exercised in the k^{th} stage, the digital output of the k^{th} stage transitions from a value equal to $f(k, comp_k^i, L)$ to a value equal to $f(k, comp_k^i, R)$. Furthermore, everytime the same comparator is exercised in a stage, the residue of this stage, which is the analog input to the following stages always transitions between the same two values. This implies that every time the i^{th} comparator is exercised in the k^{th} stage, the digital output of the x^{th} stage ($x = k+1, \cdots, N$), is always equal to the value $f(x, comp_k^i, L)$ before the transition and to the value $f(x, comp_k^i, R)$ after the transition.

We define:

$$L_i^k = [f(k, comp_k^i, L), f(k+1, comp_k^i, L), \cdots, f(N, comp_k^i, L)] \tag{4.2}$$

$$R_i^k = [f(k, comp_k^i, R), f(k+1, comp_k^i, R), \cdots, f(N, comp_k^i, R)] \tag{4.3}$$

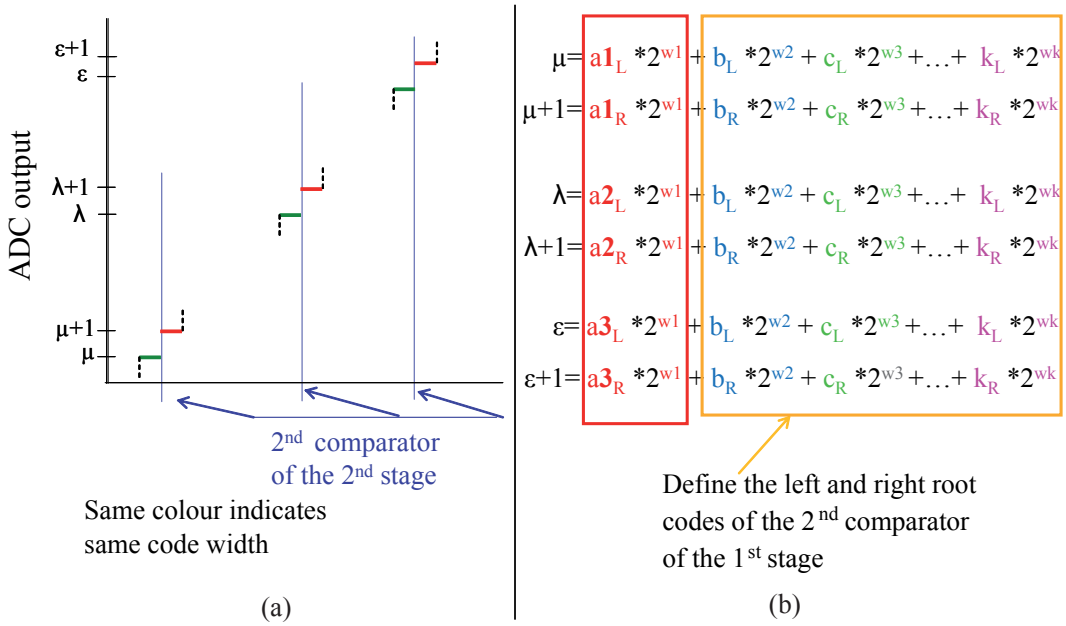


Figure 4.6: Properties of ADC output codes that are mapped to the same comparator: (a) code widths, and (b) root codes.

If we sum up the elements of L_i^k respecting the weight of each stage, then we obtain the left root code of the i^{th} comparator in the k^{th} stage. If we sum up the elements of R_i^k respecting the weight of each stage, then we obtain the right root code of the i^{th} comparator in the k^{th} stage.

In Chapter 3 we have introduced the property which consists in the fact that the ADC output codes around the transitions that are mapped to the same comparator have equal widths. The second property that we introduce here is that the ADC output codes around the transitions that are mapped to the i^{th} comparator in the k^{th} stage can be expressed on the basis of the root code of this comparator by adding to it the contribution of the previous $k-1$ stages, taking into consideration their respective weights. These two properties are shown in Figure 4.6 with an example of three ADC output transitions corresponding to the second comparator of the second stage in a k-stage pipeline ADC. The codes with the same colours are the codes which have the same width. From the property discussed in Chapter 3, we have:

$$Width_\mu = Width_\lambda = Width_\epsilon \text{ and } Width_{\mu+1} = Width_{\lambda+1} = Width_{\epsilon+1}$$

Also, if we write these codes as a function of the contribution of each of the stages, the coefficients of the stages from 2 to k will be the same for codes μ, λ, ϵ and they define the left root code of the second comparator of the second stage. Similarly, the coefficients of the stages from 2 to k will be the same for codes $\mu+1$, $\lambda+1$, $\epsilon+1$ and they define the right root code of the second comparator of the second stage.

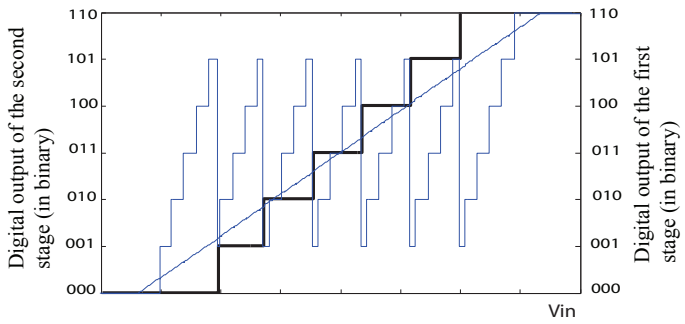


Figure 4.7: Transitions in the first and second stages.

Finally, it should be noticed that by looking at all the output codes of the ADC that are due to the i^{th} comparator in the k^{th} stage and expressing them on the basis of the root code, we find that there exist combinations of outputs of stages 1 to $k-1$ that are not present. For example, consider Figure 4.7 which superimposes the outputs of the first and second stages of the ADC. As can be seen, when the output of the first stage equals 011, only the 2^{nd}, 3^{rd}, 4^{th} and 5^{th} comparators of the second stage are exercised. This means that when calculating the ADC output codes that are due to the 1^{st} and 6^{th} comparators based on their root codes, the output 011 of stage 1 will not be taken into consideration.

4.3 Cancelling out the effect of noise

In the previous section, we have shown that the ADC output codes corresponding to the same comparator can be expressed as a function of the root code of this comparator and the contributions of the stages preceding the stage where this comparator belongs. This property can be used to cancel out the effect of the presence of noise while performing the mapping between the ADC output transitions and the comparators.

We first apply a ramp and we observe the type of the transitions in each comparator of each stage. For each natural transition of the i^{th} comparator in the k^{th} stage we obtain L_i^k and R_i^k (see Equation 7.13 and Equation 7.14). If n is the number of natural transitions, then we have at hand n values of each element $f(x, comp_k^i, L)$ of L_i^k and n values of each element $f(x, comp_k^i, R)$ of R_i^k, $x = k, \cdots, N$.

Due to the presence of noise, the extracted values for the elements of L_i^k and R_i^k for $x \geq k+1$ are not necessarily the same for each natural transition of the same comparator. In other words, the left and right root codes calculated starting from different natural transitions may not be the same.

For $x = k+1, \cdots, N$, let $\mu_{comp_k^i}^{x,L}$ and $\mu_{comp_k^i}^{x,R}$ be the values of $f(x, comp_k^i, L)$ and $f(x, comp_k^i, R)$, respectively, that are most frequently met in the n values that we have at hand. In this way we obtain the noise-free:

$$L_i^k = [i-1, \mu_{comp_k^i}^{k+1,L}, \cdots, \mu_{comp_k^i}^{N,L}], \tag{4.4}$$

$$R_i^k = [i, \mu_{comp_k^i}^{k+1,R}, \cdots, \mu_{comp_k^i}^{N,R}]. \tag{4.5}$$

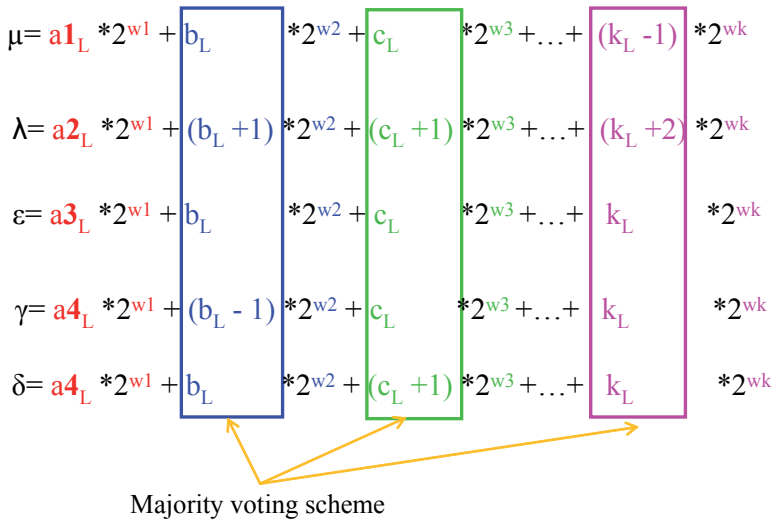


Figure 4.8: Extracting the noise free left root code example.

From the noise free L_i^k and R_i^k we can calculate the noise-free left and right root codes of the i^{th} comparator in the k^{th} stage.

In Figure 4.8 we show an example with 5 ADC output codes corresponding to the ADC output to the left of 5 transitions of one of the comparators of a second pipeline stage. The coefficients of the first stage (a_{1L}, a_{2L}, a_{3L}, a_{4L}, a_{5L}) will change depending on the position of the transition in the dynamic range. As for the coefficients of the stages 2 to k, they can only change due to noise. We apply a majority voting scheme to extract the noise free value. Thus, b_L is chosen for the second stage, c_L is chosen for the third stage, and k_L is chosen for the k^{th} stage.

It should be noticed that the number of natural transitions of a specific comparator varies depending on the number of the stage the comparator belongs to. For example, for an 11-bit, 2.5 bit/stage ADC, for one sweep of the input ramp covering the overall dynamic range, the number of natural transitions varies from 1 if the comparator is in the first stage to more than 500 if the comparator is in the fifth stage. As we show in Figure 4.2 and Figure 4.3, the noise becomes evident for the transitions of the last stages in the pipeline, thus for these stages we have a large number of values to obtain the noise-free L_i^k and R_i^k and, thereby, noise-free left and right root codes.

4.4 Obtaining the mapping

Figure 4.9 shows a snapshot of transitions of three consecutive stages, it will be used to visualize the explanations that will follow. Let lrc_k^i and rrc_k^i denote the left root code and right root code, respectively, of the i^{th} comparator in the k^{th} stage. Let also w_k denote the weight of the k^{th} stage. In the first stage, the left and right root codes of

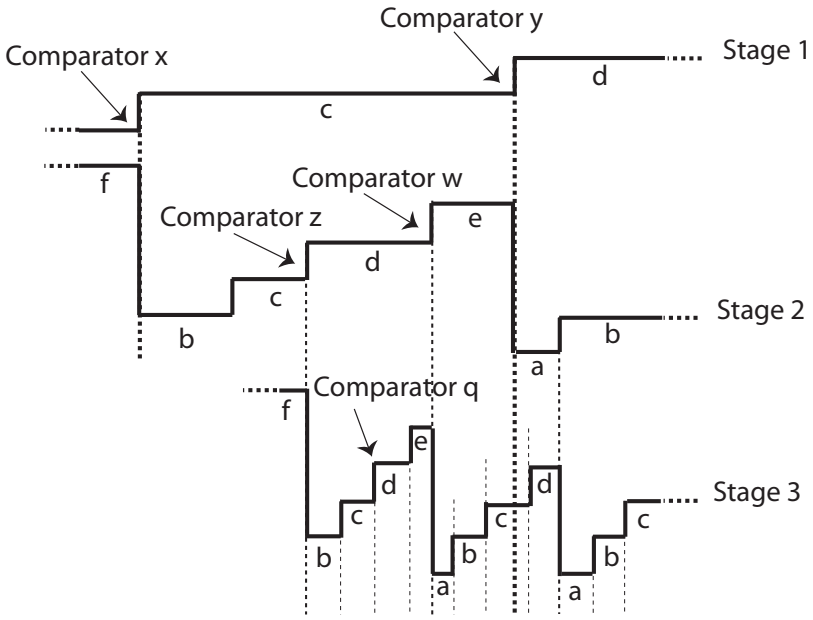


Figure 4.9: Snapshot of transitions of the first three stages.

the comparators are in fact output codes of the ADC. For example, referring to Figure 4.9, the ADC output codes lrc_1^x and rrc_1^x are mapped to comparator x of stage 1 and the ADC output codes lrc_1^y and rrc_1^y are mapped to comparator y of stage 1.

To complete the mapping, we need to find for a comparator in the k^{th} stage, $k \geq 2$, which combinations of outputs of stages 1 to $k-1$ should be added to its root codes to calculate the corresponding ADC output codes.

Consider two consecutive transitions of the k^{th} stage, where $k \geq 1$. This pair of transitions could be (natural, natural), (natural, forced), or (forced, natural). Between these two transitions the output of stages x ($x = 1, \cdots, k$), is fixed at a value Y_x while the digital output of the $(k+1)^{th}$ stage ramps from Y_{k+1} to $Y_{k+1} + t$, where t is the number of comparators that have been exercised. Referring to Figure 4.9, between two consecutive natural transitions of stage 1 that are due to comparators x and y, the digital output of stage 1 is fixed to c, the digital output of stage 2 ramps from code b to code e, and $t = 3$.

If the i^{th} comparator in the $(k+1)^{th}$ stage is exercised when its output changes from Y_{k+1} to $Y_{k+1}+t$, then the two ADC output codes $[lrc^i_{(k+1)} + Y_k \cdot 2^{w_k} + \cdots + Y_1 \cdot 2^{w_1}]$ and $[rrc^i_{(k+1)} + Y_k \cdot 2^{w_k} + \cdots + Y_1 \cdot 2^{w_1}]$ are mapped to the i^{th} comparator in the $(k+1)^{th}$ stage.

Referring to Figure 4.9, the ADC output codes $[lrc_2^z + c \cdot 2^{w_1}]$ and $[rrc_2^z + c \cdot 2^{w_1}]$ are mapped to comparator z of stage 2 and the ADC output codes $[lrc_2^w + c \cdot 2^{w_1}]$ and $rrc_2^w + c \cdot 2^{w_1}]$ are mapped to comparator w of stage 2. Between two consecutive natural transitions of stage 2 that are due to comparators z and w, the output of stage 1 is

fixed at c, the output of stage 2 is fixed at d and stage 3 ramps from code b to code e. The ADC output codes $[lrc_3^q + c \cdot 2^{w_1} + d \cdot 2^{w_2}]$ and $[rrc_3^q + c \cdot 2^{w_1} + d \cdot 2^{w_2}]$ are mapped to comparator q of stage 3.

It remains to find the values Y_{k+1} and $Y_{k+1} + t$. If the first transition of the aforementioned pair of consecutive transitions of the k^{th} stage is a natural transition of the j^{th} comparator in this stage, then Y_{k+1} equals the second element of R_j^k, which is $f(k+1, comp_j^k, R)$. Otherwise if it is a forced transition due to a natural transition of comparator h in the $(k-1)^{th}$ stage, then Y_{k+1} equals the third element of R_h^{k-1}, which is $f(k+2, comp_h^{k-1}, R)$.

Similarly, if the second transition is a natural transition of the j^{th} comparator in this stage, then $Y_{k+1} + t$ equals the second element of L_j^k, which is $f(k+1, comp_j^k, L)$. Otherwise if it is a forced transition due to a natural transition of comparator h in the $(k-1)^{th}$ stage, then $Y_{k+1} + t$ equals the third element of L_h^{k-1}, which is $f(k+2, comp_h^{k-1}, L)$.

Referring to Figure 4.9, suppose that we are considering the interval between the natural transition in stage 2 that is due to comparator w and the forced transition due to a transition of comparator y in stage 1. The output of the third stage ramps from Y_3 to $Y_3 + t$, where Y_3 equals the second element of $R_w^2 = [e, a, \cdots]$ and $Y_3 + t$ equals the third element of $R_y^1 = [d, a, c, \cdots]$.

So far we considered a pair of two consecutive transitions of the k^{th} stage. As for the range before the very first transition of the k^{th} stage, Y_{k+1} should be taken equal to the output of the $(k+1)^{th}$ stage at the beginning of the dynamic range. Similarly, regarding the range after the very last transition of the k^{th} stage, $Y_{k+1} + t$ should be taken equal to the output of the $(k+1)^{th}$ stage at the end of the dynamic range.

We have implemented an algorithm that calculates all the ADC output codes based only on the extracted L_i^k and R_i^k of each comparator. The algorithm is based on nested for loops. We explain next the implementation.

Let's start from the first pipeline stage. We know that the left and right root codes of the comparators of the first stage are ADC output codes and no calculation is needed to do the mapping for these comparators. If the digital output of the first stage of the ADC starts at a value of '0' and ends at a value of '6', this means that 6 comparators have been exercised in the first stage. We can imagine that the form of the digital output of the first stage will look as shown in Figure 4.10 (a).

Proceeding now with the second stage, we have only at hand the L_i^k and R_i^k (from which the root codes can be easily calculated). The form of the digital output of the second stage can be fully guessed from the second elements of L_i^1 and R_i^1 of the comparators of the first stage. We also know the values of the digital output of the second stage at the beginning and at the end of the dynamic range. Let's suppose they are equal to '0' and '6', respectively. At every transition of comparator i in the first stage we know the value of the digital output of the second stage on the left and on the right of this transition. For example, if we have:

$f(2, comp_1^1, L) = 6;\ f(2, comp_1^1, R) = 1;$
$f(2, comp_1^2, L) = 5;\ f(2, comp_1^2, R) = 1;$

$f(2, comp_1^3, L) = 6;\ f(2, comp_1^3, R) = 0;$
$f(2, comp_1^4, L) = 6;\ f(2, comp_1^4, R) = 0;$
$f(2, comp_1^5, L) = 5;\ f(2, comp_1^5, R) = 1;$
$f(2, comp_1^6, L) = 6;\ f(2, comp_1^6, R) = 1.$

This implies having information about the form of the digital output of the second stage as shown in Figure 4.10 (b). From this we can easily guess the complete form of the digital output of the second stage as it will ramp from '0' to $f(2, comp_1^1, L) = 6$ when the digital output of the first stage is constant at '0' and from $f(2, comp_1^6, R) = 1$ to '6' when the digital output of the first stage is constant at '6'. As for the intermediate values of the first stage, the digital output of the second stage will ramp from $f(2, comp_1^i, R)$ to $f(2, comp_1^{i+1}, L)$ between the transitions of $comp_1^i$ and $comp_1^{i+1}$. The complete form of the digital output of the second stage is shown Figure 4.10 (c).

In this way, we are able to guess which are the comparators that are being exercised in the second stage and which are the contributions of the first stage that should be added to the root codes of each comparator. For example, referring to Figure 4.10(c), for the sixth comparator of the second stage, the contributions of the first stage that should be considered to be added to its root codes are only 2, 3, 5 and 6. This comparator does not get exercised while the output of the first stage is equal to 1 and 4.

Similarly, after *reconstructing* the digital output of the first and second stages we can deduce the form of the digital output of the third stage by relying only on the extracted $L_i^1(3)$,$R_i^1(3)$ and $L_i^2(2)$,$R_i^2(2)$. When the digital output of the first stage is equal to '0', the digital output of the second stage ramps from '0' to '6'. In order to build the digital output of the third stage corresponding to this zone, we apply the same reasoning as we did when constructing the second stage digital output when the digital output of the first stage ramped from '0' to '6'. The same applies for the other values of the digital output of the first stage.

For the regions where there is a forced transition, we have to consider the $f(3, comp_1^i, L)$ and $f(3, comp_1^i, R)$ in order to know where to stop considering a contribution of the first stage equal to $f(1, comp_1^i, L)$ and start considering a contribution of the first stage equal to $f(1, comp_1^i, R)$. We show this in the example of Figure 4.11. The comparators of the third stage that are exercised in the region referred to as **Z1** will have a contribution of the first stage equal to 3 and a contribution of the second stage equal to 5. In the region referred to as **Z2** the comparators of the third stage will have a contribution of the first stage equal to 3 and a contribution of the second stage equal to 6 before the natural transition of the first stage. After the natural transition of the first stage, the contribution of the first stage will be equal to 4 and the contribution of the second stage will be equal to 0.

In this way, we are able to *reconstruct* the digital output of the third stage and we are able to deduce which are the comparators that are being exercised in the third stage and which are the contributions of the first and second stages that should be added to the root codes of each comparator in the third stage.

The reasoning continues in this way down to the last stage of the pipeline ADC. Note that in the digital output of the fourth stage there are going to be forced transitions

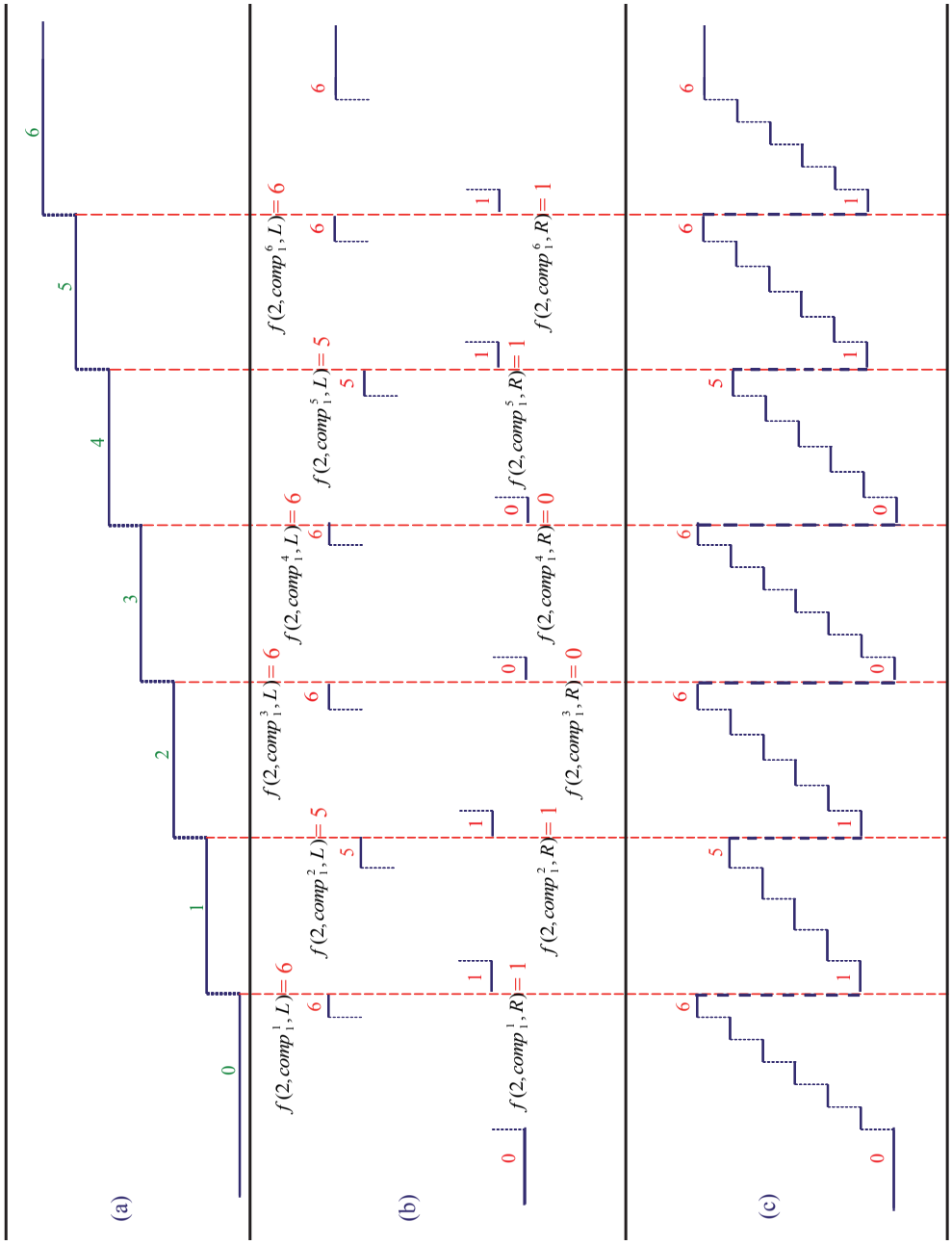


Figure 4.10: Reconstructing the digital output of the second stage from $L_i^1(2)$ and $R_i^1(2)$.

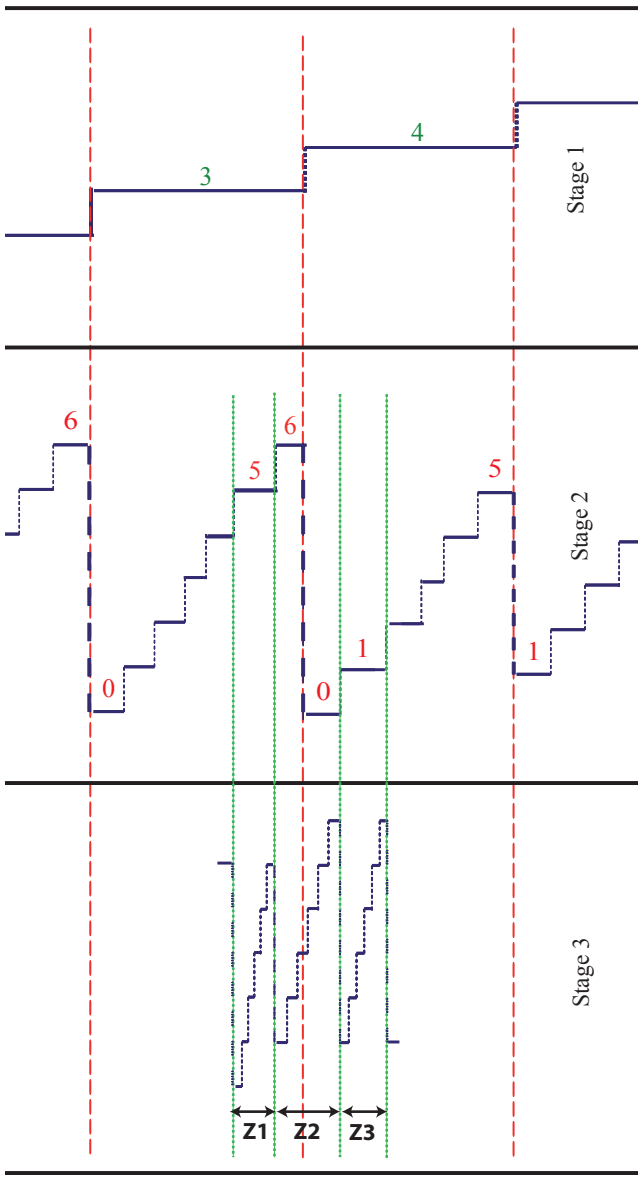


Figure 4.11: Forced transitions regions.

due to the first, second and third stages. In the fifth stage there are going to be forced transitions due to the first, second, third and fourth stages.

We implemented the algorithm using nested for loops. The underlying principle is to start from the first stage and for every output value Y_k of stage k calculate the ADC output codes corresponding to the comparators of stage $k+1$ that are exercised when the digital output of stage $k+1$ ramps from:

1. $f(k+1, comp_k^a, R)$ to $f(k+1, comp_k^b, L)$ in the case where stage k transitions to an output value Y_k due to the natural transition of $comp_k^a$ and transitions from Y_k to another output value due to the natural transition of $comp_k^b$.

2. $f(k+1, comp_k^a, R)$ to $f(k+1, comp_x^b, L)$, $x = 1, \cdots, k-1$ in the case where stage k transitions to an output value Y_k due to the natural transition of $comp_k^a$ and transitions from Y_k to another output value due to the natural transition of $comp_x^b$, $x = 1, \cdots, k-1$.

3. $f(k+1, comp_x^a, R)$ to $f(k+1, comp_k^b, L)$, $x = 1, \cdots, k-1$ in the case where stage k transitions to an output value Y_k due to the natural transition of $comp_x^b$, $x = 1, \cdots, k-1$. and transitions from Y_k to another output value due to the natural transition of $comp_k^b$.

4. *Output of the stage* $k+1$ *in the beginning of the dynamic range* to $f(k+1, comp_k^1, L)$ in the case where the output value Y_k of stage k is its output value at the beginning of the dynamic range.

5. $f(k+1, comp_k^6, R)$ to *Output of the stage* $k+1$ *at the end of the dynamic range* in the case where the output value Y_k of stage k is its output value at the end of the dynamic range.

The reader can imagine how the algorithm runs for mutiple stages. Below is given the Matlab code with 5 nested for loops which calculates for a 5 stage pipeline ADC (four 2.5-bit stages and a last 3-bit stage) all the ADC output codes for comparators which fit in scenario (1) described above. In order to take into consideration the scenarios from (2) to (5) we use the same principle and we change the limits of the for loop, and use 4, 3 or 2 nested for loops in order to be able to calculate all ADC output codes.

```
% j1, j2, j3, j4 and j5 refer to digital output of stages 1, 2, 3, 4 and 5.
% h2, h3, h4 and h5 are 2-D matrices of the indices of the calculated ADC output codes mapped to comparators of stage 2, 3, 4 and 5
% Code_Right and Code_Left are f(k+1,comp(k,i),R) and f(k+1,comp(k,i),L) of stages 1 to k-1
% code_second_left and code_second_right are matrices of ADC output codes mapped to the comparators of stage 2
% code_third_left and code_third_right are matrices of ADC output codes mapped to the comparators of stage 3
% code_fourth_left and code_fourth_right are matrices of ADC output codes mapped to the comparators of stage 4
```

```
% code_fifth_left and code_fifth_right are matrices of ADC output codes
    mapped to the comparators of stage 5
% Root_second_left and Root_second_right are matrices of the root codes
    for the comparators of stage 2
% Root_third_left and Root_third_right are matrices of the root codes for
    the comparators of stage 3
% Root_fourth_left and Root_fourth_right are matrices of the root codes
    for the comparators of stage 4
% Root_fifth_left and Root_fifth_right are matrices of the root codes for
    the comparators of stage 5

for j1=1:5
    for j2=(Code_Right(1,j1)+1):(Code_Left(1,(j1+1))-1)
        h2(j2)=h2(j2)+1;
        code_second_left(j2,h2(j2))=Root_second_left(j2)+j1*2^8;
        code_second_right(j2,h2(j2))=Root_second_right(j2)+j1*2^8;

        for j3=(Code_Right(2,j2)+1):(Code_Left(2,(j2+1))-1)
            h3(j3)=h3(j3)+1;
            code_third_left(j3,h3(j3))=Root_third_left(j3)+j1*2^8+j2*2^6;
            code_third_right(j3,h3(j3))=Root_third_right(j3)+j1*2^8+j2
                *2^6;

            for j4=(Code_Right(3,j3)+1):(Code_Left(3,(j3+1))-1)
                h4(j4)=h4(j4)+1;
                code_fourth_left(j4,h4(j4))=Root_fourth_left(j4)+j1*2^8+
                    j2*2^6+j3*2^4;
                code_fourth_right(j4,h4(j4))=Root_fourth_right(j4)+j1
                    *2^8+j2*2^6+j3*2^4;

                for j5=(Code_Right(4,j4)+1):(Code_Left(4,(j4+1))-1)
                    h5(j5)=h5(j5)+1;
                    code_fifth_left(j5,h5(j5))=Root_fifth_left(j5)+j1
                        *2^8+j2*2^6+j3*2^4+j4*2^2;
                    code_fifth_right(j5,h5(j5))=Root_fifth_right(j5)+j1
                        *2^8+j2*2^6+j3*2^4+j4*2^2;

                end;
            end;
        end;
    end;
end;
```

4.5 Summing up

To sum up, referring to the steps of the reduced code testing technique that were listed at the end of section 3.5 of Chapter 3, the main challenge of the technique is in step 1. This step consists in finding the mapping between the ADC output transitions and the comparators that are being exercised. In Chapter 3 we have proposed to carry out the mapping by monitoring the transitions that are happening at the output of each stage. In Section 4.1 of this Chapter we have shown what might go wrong in the presence of noise. Afterwards, we have explained the root codes property in pipeline

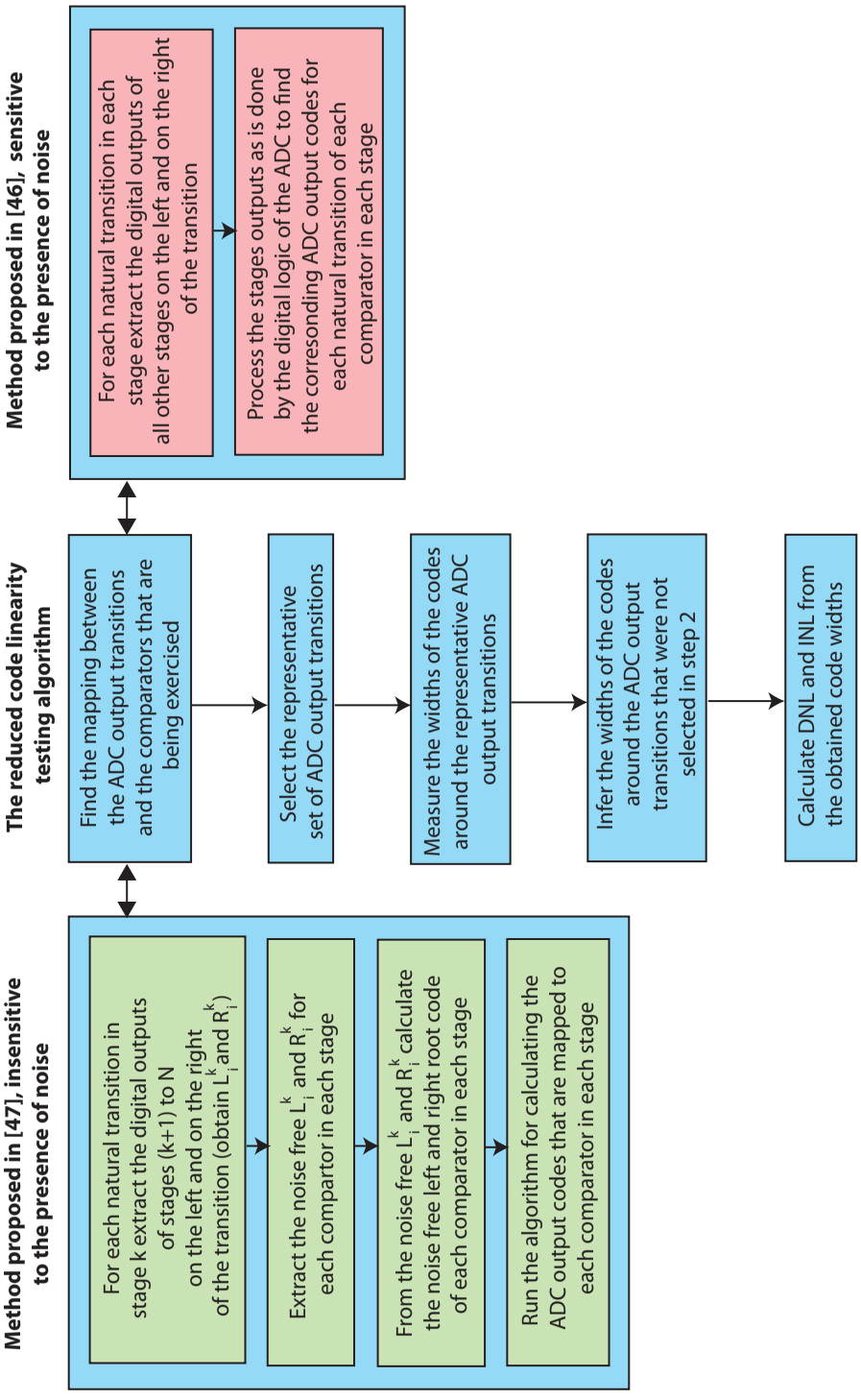


Figure 4.12: The steps of the reduced code testing technique.

ADCs and we have proposed a noise cancelling scheme which exploits this property. This will result in a very accurate mapping even in the presence of noise as will be shown with experimental data in the next Chapter. Figure 4.12 resumes the steps of the reduced code testing technique and the mapping methods that we have proposed, including the first method that is noise sensisitive [46] and the ultimate technique that is noise-insensitive [47].

Chapter 5

Experimental results

5.1 The test setup

In the following we will show experimental results which validate the theory of the techniques that have been proposed in the previous Chapters. Our case study is an 11-bit, 55nm pipeline ADC from STMicrolectronics. This ADC is composed of four 2.5-bit stages and a last 3-bit stage. For this particular ADC, we had access to the digital output of the internal stages before digital correction is applied (this makes a total of 15 bits). Thus, we extracted the digital output of the internal stages off-chip and we did the data processing in Matlab. Figure 5.1 shows a photo of the test board and Figure 5.2 shows a photo of the test setup (input stimulus generator not shown).

We used a sinewave as input stimulus. We did a classical sinusoidal histogram test to compare with our technique. The number of samples to be acquired for a sinusoidal histogram test depends on the resolution of the ADC and on the type of sampling (coherent or random). Table 5.1 compares the number of samples that are needed to perform the test depending on the type of sampling and on the resolution of the ADC [48].

The outputs of the stages were acquired before digital correction is performed. In this case the digital outputs of the stages are concatenated without making a 1-bit overlap. We can see 6 jumps in the signal shown in Figure 5.3 corresponding to the six comparators of the first stage. In order to perform a classical histogram test we

Table 5.1: Number of samples to be acquired for a sinusoidal histogram test.

N	Random	Coherent
8	67000	4022
10	267 500	16 085
12	1 050 000	64 340
14	4 282 500	257 360
16	17 125 000	1 029 437

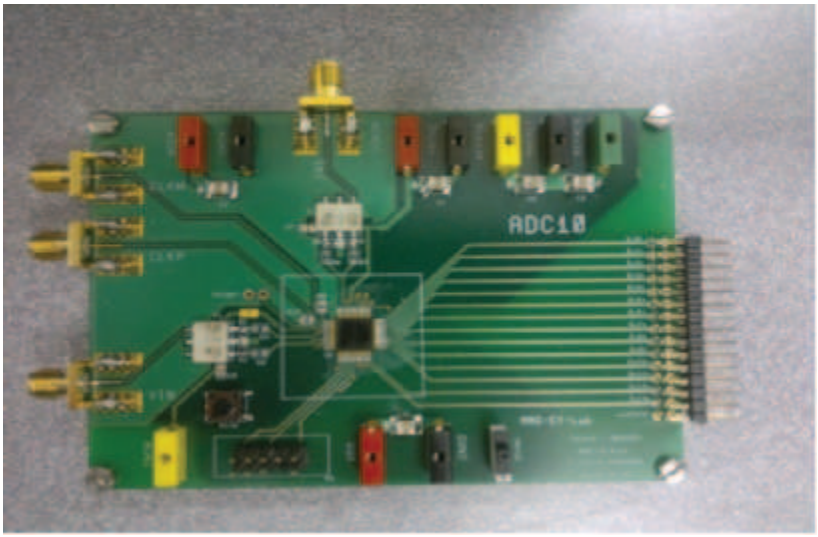


Figure 5.1: Photo of the testboard.

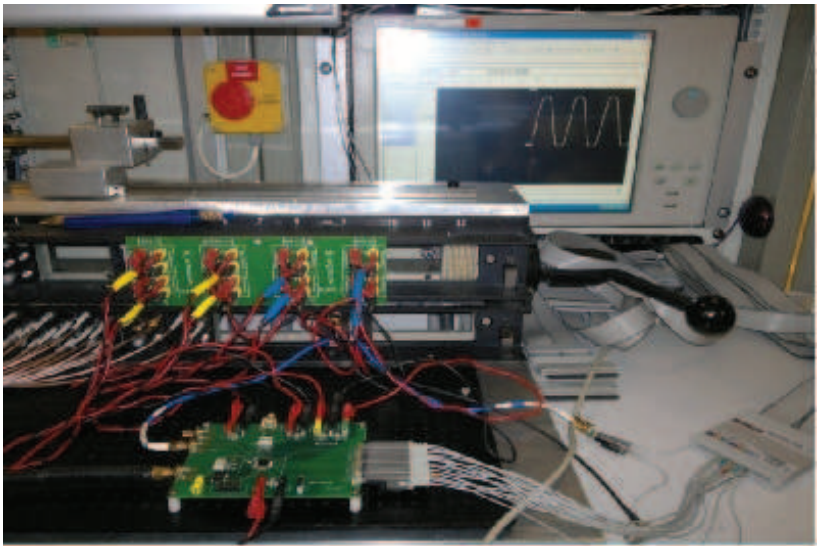


Figure 5.2: Photo of the experimental set-up.

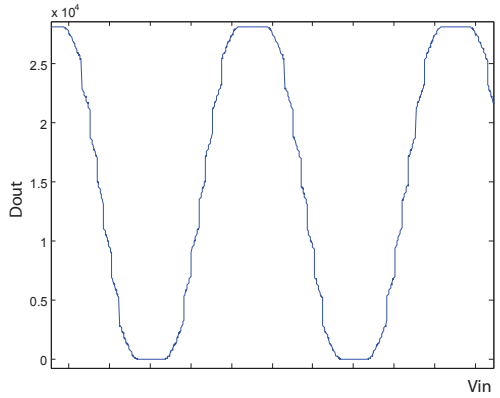


Figure 5.3: Acquired digital outputs.

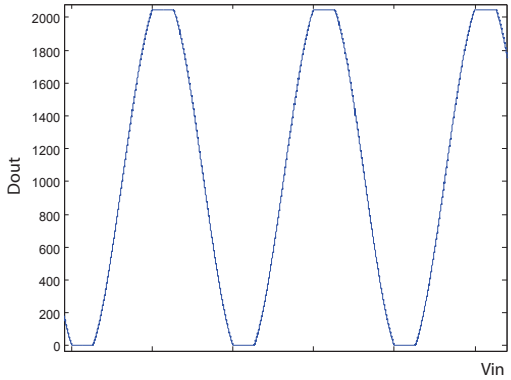


Figure 5.4: After performing digital correction in Matlab.

need to reconstruct the output of the ADC by processing the output of the stages as it might have been done in a digital logic block of this ADC. This results in a quantized sinusoidal signal as in shown in Figure 5.4.

If we split the acquired 15 bits in groups of 3 bits, where each group corresponds to each of the stages of the ADC we get the digital outputs of the stages, as shown in Figure 5.5. Note that the digital output of stages 1 through 4 varies from 0 to 6 (000 to 110 in binary) as they are 2.5-bit stages. The digital output of the last stage varies from 0 to 7 (000 to 111 in binary) as it is a 3-bit stage. If we superimpose the outputs of all the stages and zoom in we obtain a plot as shown in Figure 4.3, which has been discussed in the previous chapter.

5.2 Verification of the basic principles

First, we verified the first property of the technique which states that the codes related to the ADC output transitions that involve the same comparator have equal DNLs. Consider Figure 5.6 which superimposes the digital output of the complete ADC on the digital output of the second stage as we traverse the whole dynamic range. As

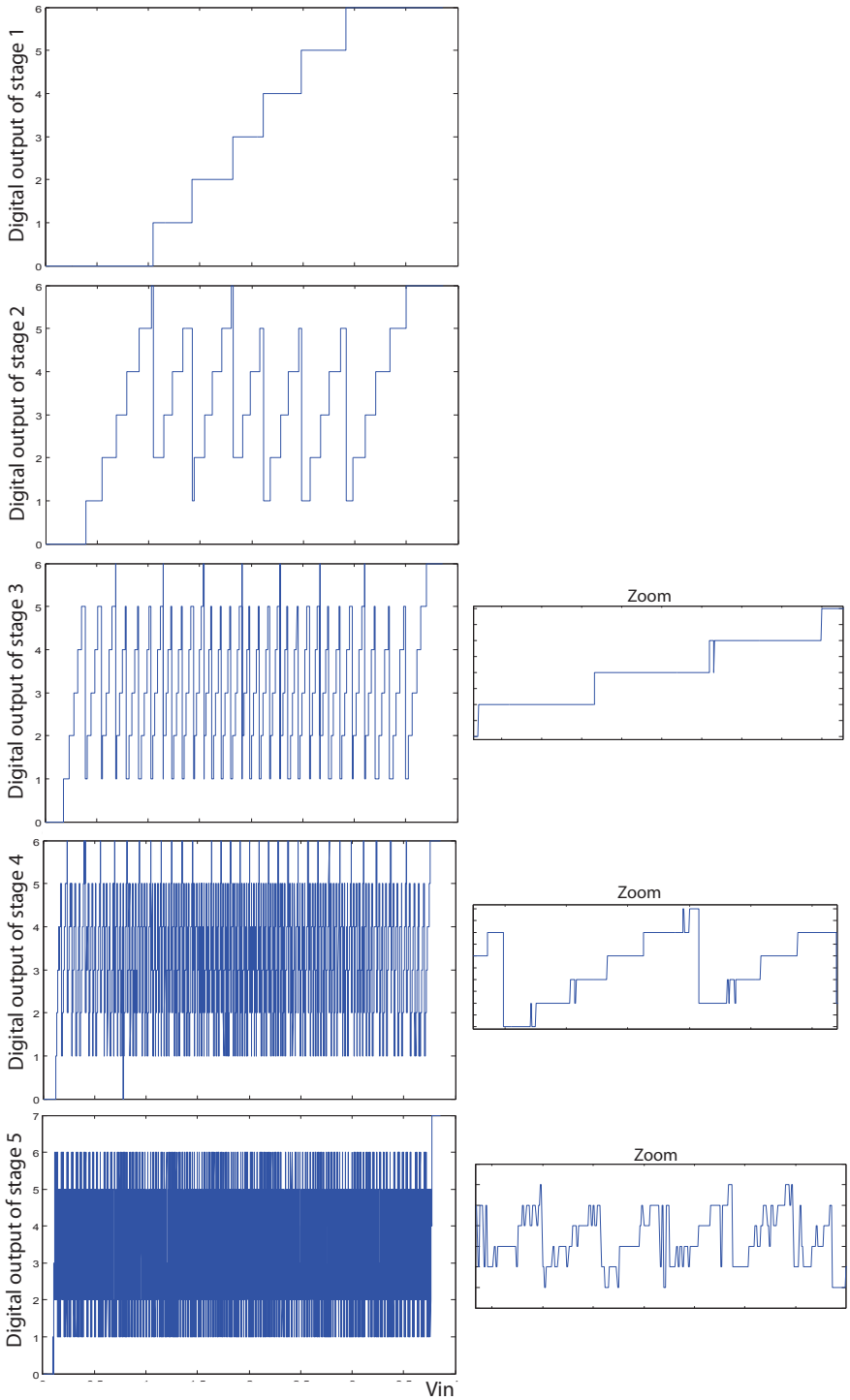


Figure 5.5: Stage outputs.

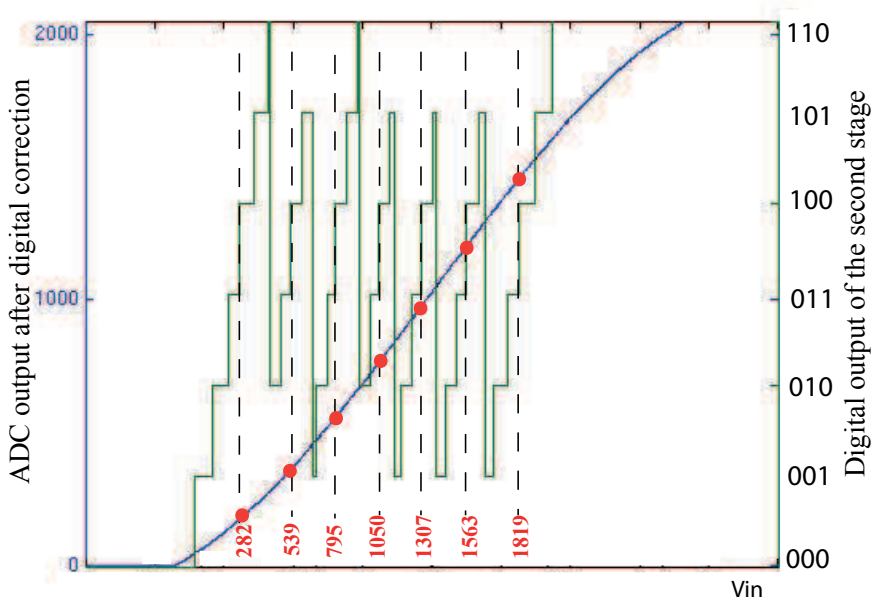


Figure 5.6: Searching for the codes that are mapped to the fourth comparator in the second stage.

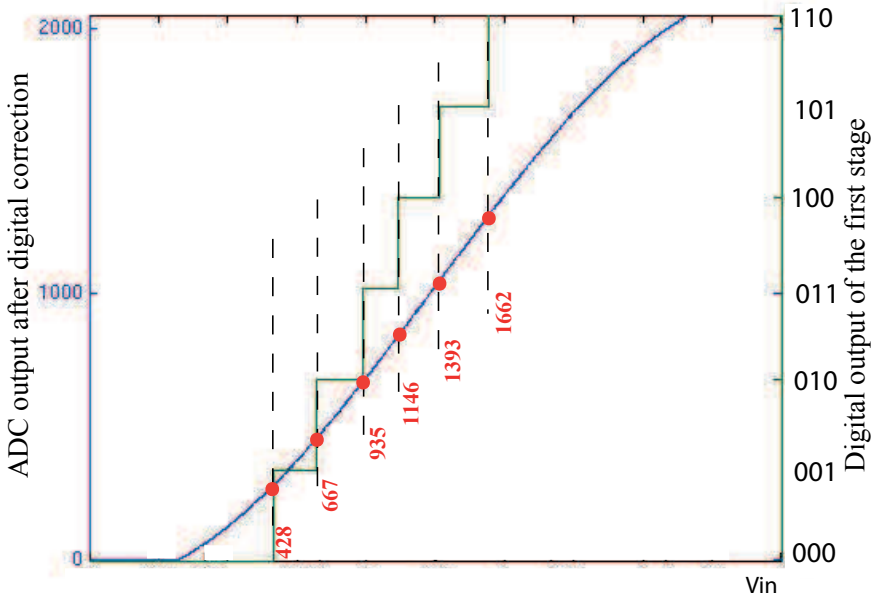


Figure 5.7: Searching for the codes that are mapped to the comparators of the first stage.

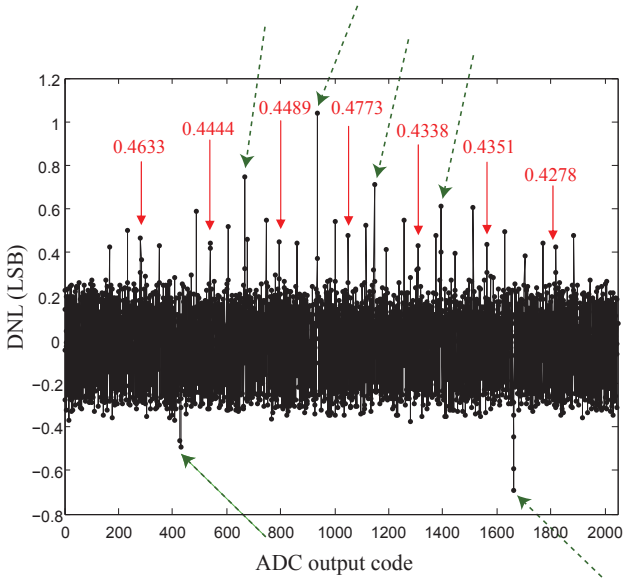


Figure 5.8: DNL obtained with the standard histogram technique. The vertical arrows point to the codes that are mapped to the fourth comparator in the second stage, while the inclined green arrows point to the codes that are mapped to the comparators of the first stage.

an example, the intersection points between the dashed vertical lines and the digital output of the complete ADC correspond to the codes that are mapped to the transitions that involve the fourth comparator in the second stage (e.g., transition from 011 to 100 labelled as C4 in Table 3.1). The codes on the right of these transitions are shown in the bottom of Figure 5.6. The first transition is mapped to code 282, the second transition is mapped to code 539, the third transition is mapped to code 795, etc.

Next, the DNL of all the codes was calculated using the standard histogram technique, as shown in Figure 5.8. By the vertical arrows we point to the DNL of the codes in Figure 5.6. As can be seen, they are very close to each other (they vary between 0.4278 and 0.4773) which confirms the basic principle of the technique. Thus, instead of measuring the DNL of all these seven codes, we can choose to measure the DNL of just one code and then use this DNL value for the rest of the codes.

Next, we verified another principle of the technique which states that the largest DNL errors happen around the output transitions that involve the comparators in the stages that are towards the front of the pipeline. As an example, Figure 5.7 shows the digital output of the first stage superimposed on the digital output of the complete ADC. The six codes shown in the bottom of Figure 5.7 are the codes positioned on the right of the transitions of the first stage. In Figure 5.8, the position of these codes is shown with the inclined dashed green arrows. As can be seen, these codes present the maximum and minimum DNL errors which justifies why we should avoid selecting a representative ADC output transition for a certain comparator that involves in addition to this comparator a comparator in one of the previous stages. It also justifies why we

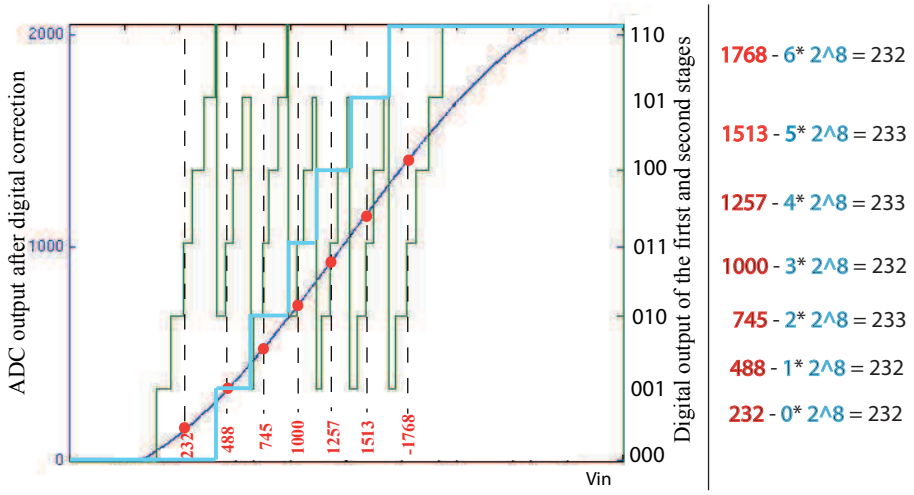


Figure 5.9: Root codes example 1.

should give priority to the first stages in the step where we extract the unmeasured code widths from the measured ones.

Next, we verified the root codes property that we have dissussed in Chapter 4. Figure 5.9 superimposes the digital output of the complete ADC on the digital output of the second stage and the first stage as we traverse the whole dynamic range. As an example, the intersection points between the dashed vertical lines and the digital output of the complete ADC correspond to the codes that are mapped to the transitions that involve the third comparator in the second stage (e.g., transition from 010 to 011 labelled as C3 in Table 3.1). The codes on the right of these transitions are shown in the bottom of Figure 5.9. The first transition is mapped to code 232, the second transition is mapped to code 488, the third transition is mapped to code 745, etc. Let's take these codes (232, 488, 745, 1000, 1257, 1513 and 1768) and substract from them the contribution of the first stage times the weight of the first stage. The weight of the first stage in this ADC is equal to 2^8. For example, in the region of the transition corresponding to the output code 745 the digital output of the first stage is equal to 2. We calculate ($ADC\ output\ code - Contribution\ of\ the\ first\ stage * 2^8$), this gives:

$(1768 - 6 * 2^8 = 232)$
$(1513 - 5 * 2^8 = 233)$
$(1257 - 4 * 2^8 = 233)$
$(1000 - 3 * 2^8 = 232)$
$(745 - 2 * 2^8 = 233)$
$(488 - 1 * 2^8 = 232)$
$(232 - 0 * 2^8 = 232)$

As expected, we obtain the same root value and due to the presence of noise it is toggling between 232 and 233. 232 seems to be more frequent and thus it defines

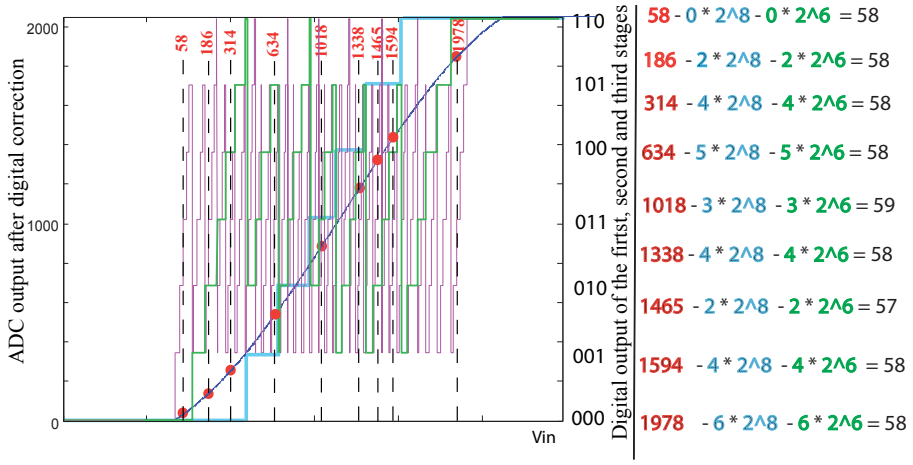


Figure 5.10: Root codes example 2.

the right root code of the third comparator of the second stage. This means that the codes that should be grouped together as having the same DNLs are not (232, 488, 745, 1000, 1257, 1513 and 1768) as it would have been suggested without using the noise cancelling scheme, but (232, 488, 744, 1000, 1256, 1512 and 1768). These are the codes that are mapped to this comparator recalculated based on the same root code (232) by addding the different contributions of the first stage.

In Figure 5.10 we give another example where we superimpose the digital output of the complete ADC on the digital output of the third, second and first stages as we traverse the whole dynamic range. As an example, the intersection points between the dashed vertical lines and the digital output of the complete ADC correspond to the codes that are mapped to the transitions that involve the third comparator in the third stage (e.g., transition from 010 to 011 labelled as C3 in Table 3.1). The codes on the right of some of these transitions are shown at the top of Figure 5.10. The first transition is mapped to code 58, the second transition is mapped to code 186, the third transition is mapped to code 314, etc. Let's take the codes (58, 186, 314, 634, 1018, 1082, 1338, 465, 1594 and 1978) and substract from them the contribution of the first stage times the weight of the first stage and also the contribution of the second stage times the weight of the second stage. The weight of the first stage in this ADC is equal to 2^8 and the weight of the second stage is equal to 2^6. For example, in the region of the transition corresponding to the output code 634 the digital output of the first stage is equal to 2 and the digital output of the second stage is equal to 5. We calculate $(ADC\ output\ code - Contribution\ of\ the\ first\ stage * 2^8 - Contribution\ of\ the\ second\ stage * 2^6)$, this gives:

$(1978 - 6 * 2^8 - 6 * 2^6 = 58)$
$(1594 - 5 * 2^8 - 4 * 2^6 = 58)$
$(1465 - 5 * 2^8 - 2 * 2^6 = 57)$
$(1338 - 4 * 2^8 - 4 * 2^6 = 58)$

$(1082 - 3 * 2^8 - 4 * 2^6 = 58)$
$(1018 - 3 * 2^8 - 3 * 2^6 = 59)$
$(634 - 1 * 2^8 - 5 * 2^6 = 58)$
$(314 - 0 * 2^8 - 4 * 2^6 = 58)$
$(186 - 0 * 2^8 - 2 * 2^6 = 58)$
$(58 - 0 * 2^8 - 0 * 2^6 = 58)$

As expected, we obtain the same root value 58 and due to the presence of noise it has once taken a value of 57 and once the value 59. 58 defines the right root code of the third comparator of the third stage. This means that the codes that should be grouped together as having the same DNLs are not (1978, 1594, 1465, 1338, 1257, 1082, 1018, 634, 314, 186 and 58) as it would have been suggested without using the noise cancelling scheme, but (1978, 1594, 1466, 1338, 1257, 1082, 1017, 634, 314, 186 and 58). These are the codes that are mapped to this comparator recalculated based on the same root code (58) by addding the corresponding contributions of the first and second stages.

5.3 DNL and INL results

After verifying the principles of the technique, we calculated the DNL and INL based on a reduced set of codes. We considered:

- 8 codes around each of the six ADC output transitions representing the six comparators of the first stage,
- 6 codes around each of the six ADC output transitions representing the six comparators of the second stage,
- 4 codes around each of the six ADC output transitions representing the six comparators of the third stage,
- 2 codes around each of the six ADC output transitions representing the six comparators of the fourth and fifth stages.

In total, we rely only on $8 \times 6 + 6 \times 6 + 4 \times 6 + 2 \times 6 + 2 \times 6 = 132$ out of 2046 codes of an 11-bit ADC, that is, on only 6% of the codes, which represents a very significant code reduction.

The DNL and INL obtained using the standard histogram technique are shown in Figure 5.11(a) and Figure 5.12(a), respectively. Figure 5.11(b) and Figure 5.12(b) show the DNL and INL obtained using the reduced code testing technique in which the mapping is deduced directly from monitoring the digital output of the stages without further processing to reduce the effect of noise (using the mapping technique on the right of Figure 4.12). Figure 5.11(c) and Figure 5.12(c) show the DNL and INL obtained using the reduced code testing technique in which the mapping is obtained by taking into consideration the noise cancelling scheme (using the mapping technique on the left of Figure 4.12).

The estimated profiles of DNL are more regular since they are extracted from the DNLs of a reduced set of codes. The highest DNL errors in the ADC correspond to the first stages in which the transitions are not very noisy. Thus, the miniumm and maxiumum DNLs were equally well captured using both mapping techniques (Figure 5.11(b) and Figure 5.11(c)). When comparing the profile of the smaller DNLs in Figure 5.11(b) and Figure 5.11(c) we can observe that the DNL profile in Figure 5.11(b) is not as 'dense' as the real DNL profile (Figure 5.11(a)). This is explained by the fact that due to mapping errors there have been codes which have been attributed smaller DNL values than their real DNL values.

The estimated INL was obtained by integrating the estimated DNL. When using the mapping technique without noise cancelling (Figure 5.12(b)), we observe that the general INL profile is well captured, however, if we superpose this INL on the INL obtained with the histogram and zoom in (Figure 5.13(a)) we observe the result of the erroneous accumulation of DNLs. This stems from the fact that the succession of the transitions is not well captured in the last stages. If the ADC had smaller DNL errors, the erroneous summation of the DNLs may result in a bad INL estimation. In other words, the presence of high peaks of DNL helped define the general INL form even if the succesion of the smaller DNLs corresponding to the last, noisy stages is not very accurate.

When using the mapping technique with noise cancelling we obtain a very good INL estimation (Figure 5.12(c)). This is due to the fact that we are summing up a correct succession of DNLs. This results in an INL estimation practically indistinguishbale from the INL obtained with the histogram test. Figure 5.13(b) shows this better as we superpose this estimated INL on the INL obtained with the histogram test. The zoomed zone confirms the fact that using the noise cancelling scheme allows making very good INL estimation, independantly from the DNL values of the ADC, even in the presence of noise.

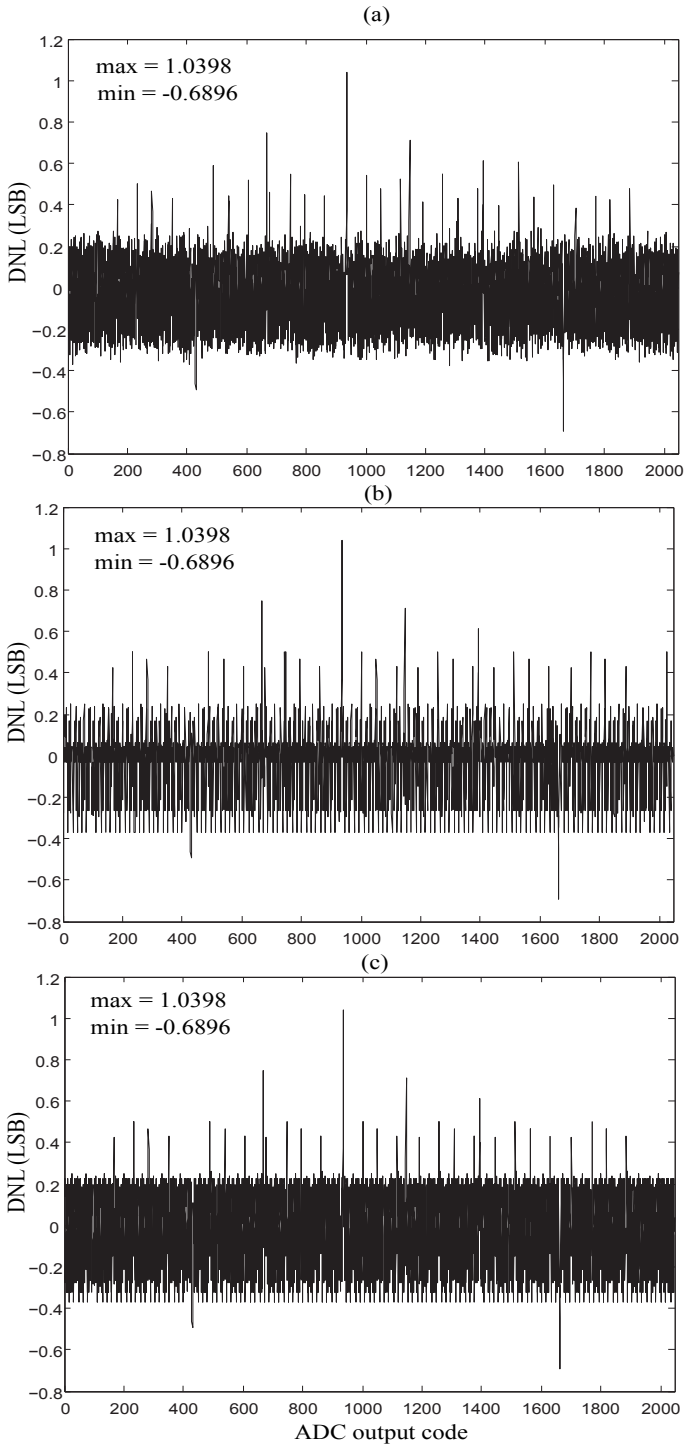


Figure 5.11: DNL obtained with: (a) the standard histogram technique; (b) reduced code testing without noise cancelling; and (c) reduced code testing with noise cancelling.

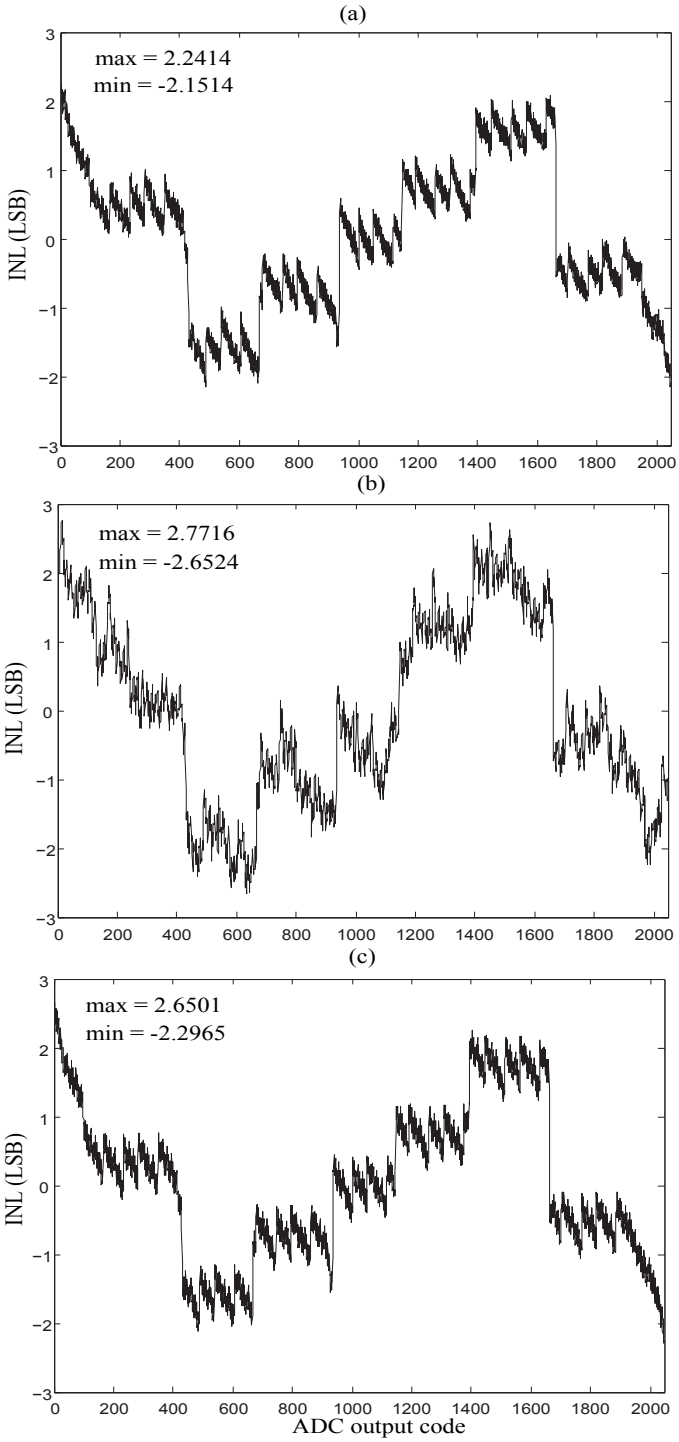


Figure 5.12: INL obtained with: (a) the standard histogram technique; (b) reduced code testing without noise cancelling; and (c) reduced code testing with noise cancelling.

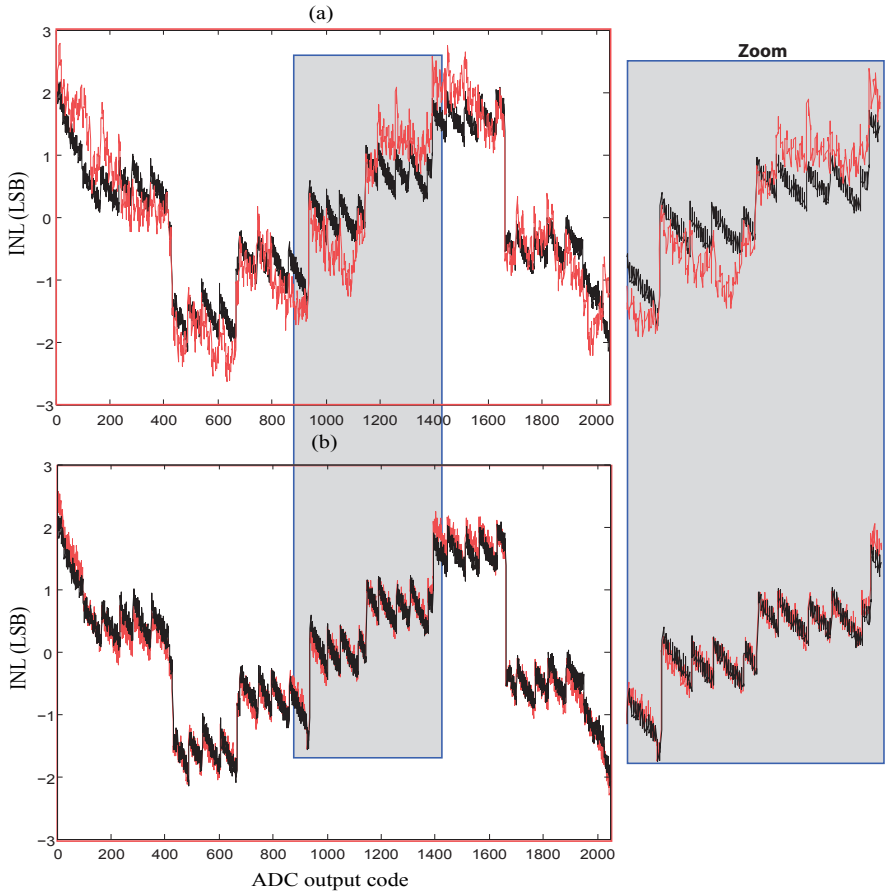


Figure 5.13: Showing the improvement in INL estimation when using the noise cancelling scheme: (a) superposition of Figure 5.12(a) and Figure 5.12(b); and (b) superposition of Figure 5.12(a) and Figure 5.12(c).

Chapter 6

Summary of contributions and perspectives

6.1 Summary of contributions

In this thesis, I analyzed thoroughly the design, architecture and functionality of pipeline ADCs, based on which I proposed techniques for efficiently applying reduced code testing on pipeline ADCs. The reduced code testing technique for pipeline ADCs is based on the fact that the ADC output transitions that are due to the same comparators have equal widths. Thus, to extract the complete transfer characteristic, it suffices to consider only a representative subset of output transitions which covers all comparators in all stages.

I have first studied the existing reduced code testing technique. After applying it to our case study and finding its shortcomings, I have proposed a new reduced code testing technique that is based on monitoring the digital outputs of the stages. The digital monitoring of the stages is efficient independently from the error sources that might be present in the pipeline ADC stages. The rationale is that when a comparator threshold is crossed, this necessarily changes the digital output of the stage, and the specific change from one code to another gives full information about which comparator has been exercised. Thus, mapping the ADC output transitions to the corresponding comparators is always correct, as opposed to the existing approach. This approach consists in producing jumps on the ADC transfer characteristic by forcing the LSB of the stages outputs to zero and than relying on an expected order of transitions to perform the mapping.

Developping this technique has required full understanding of what is happening at the digital output of the pipeline stages. To this end, I identified two types of transitions that could happen at the digital output of a pipeline stage. The first type is *natural* transitions that happen in stage k when the i^{th} comparator threshold in stage k is crossed. The ADC output codes corresponding to theses transitions can be representive of comparator i if they were not preceded by any comparator natural transition in any previous stage. The second type is $forced$ transitions that happen in a stage k under the influence of a comparator being exercised in a previous stage.

Thus, these transitions are representative of the comparator that have caused them in that previous stage (if there are many, the comparator in the stage that is closest to the beginning of the pipeline takes over). Once an ADC output transition is identified to be representative of a certain comparator in a certain stage, it can be selected so that the widths of the codes around it will be measured. Otherwise, the widths of the codes around it will be inferred later by using the measurements of the codes around another ADC output transition that is representative of the same comparator in the same stage.

The fact that the technique I proposed relies on monitoring the transitions of the digital output of the stages makes it sensitive to noise. I solved this problem by identifying another property in pipeline ADCs and using it to cancel out the inaccuracies that happen due to the presence of noise. This property has been referred to as the *root codes* property. The ADC output codes that are due to the same comparator can all be expressed on the basis of the *root code* of this comparator by adding to it the contributions of the previous stages. If we express all the ADC output codes that are due to the same comparator i in stage k as a function of the individual stages contributions, we will find the same stages contributions from stage k to N and by adding them respecting their weights we obtain the root code of comparator i in stage k. The value of this root code can be used to deduce all ADC output codes that are due to the same comparator. I used this property to cancel out the noise effect by applying a majority voting scheme when the coefficients that should be the same are found out to be different due to the presence of noise. I have implemented an algorithm that is based on nested for loops and that permits calculating all ADC output codes that are due to the same comparator from the noise free root codes.

Finally, I validated this work on an experimental 11-bit, 55nm pipeline ADC from STMicroelectronics. First, the principles of the technique and the properties that I have introduced were verified. Afterwards, DNL and INL of this ADC were calculated using the classical histogram technique. The result of the histogram was compared to the estimated DNL and INL obtained using the proposed techniques: the first one that is noise sensitive and the second one that is noise-insensitive. The results show that the estimated DNL and INL are practically indistinguishable from the DNL and INL that are obtained with the standard histogram technique, and this by using measurements of only 6% of the codes, showing significant improvement when the noise cancelling method is used.

This work can be readily used as a Design-For-Test technique to cut down the production test time of pipeline ADCs. Actually, the time consuming step in the production test of high resolution ADCs nowadays is the transfer of a huge number of histogram samples from the tester to the work station. Since only 6% of the code widths of an 11-bit ADC are necessary to be measured, this means that only 6% of the histogram data is needed and this will translate in a very significant test time reduction.

To conclude, I have shown in this thesis that a lot can be done for testability if the ADC architecture is analysed for this purpose. The collaboration between designers and test engineers should be enhanced in order to find solutions for ADC test, particularly, and for all other analog and mixed-signal circuits in general. If the circuit designers will continue building complex and high performance circuits without considering the

testability problem, then the test cost will take over the total cost of fabricating analog and mixed-signal devices. This thesis is a step towards building efficient test solutions that are architecture dependant and that I believe is the way to go in order to find efficient solutions that will solve the analog and mixed-signal circuit test problem.

6.2 Perspectives

In the literature the work that has been reported concerning reduced code test techniques have concerned so far pipeline and SAR ADCs. I have shown the limitations of the existing approach for pipeline ADCs and I have proposed a new technique. The technique proposed for SAR ADCs [34] is intended for a basic architecture of these ADCs. Nowadays, SAR ADCs include redundancy schemes [49] similar to those found in pipeline ADCs. Some of the principles that I have developped for pipeline ADCs could be applicable to more recent SAR architectures. Furthermore, the technique presented here can be readily used for sub-ranging ADCs. These ADCs are multi-step ADCs, similar to pipeline ADCs except that the number of bits in each stage can be higher than 3 or 4 bits. As for cyclic ADCs, they can be seen as a 1-stage pipeline ADC and thus the number of codes that needs to be measured could be very small. For example, if the 1-stage contains 2 comparators then only four codes need to be measured.

Apart from the flash ADC, all Nyquist-rate ADCs perform the conversion in many steps, either in hardware as in the case of pipeline and sub-ranging ADCs or in time as in the case of SAR and cyclic ADCs. Thus, if their architecture is studied in detail, they all offer the possibility to apply reduced code linearity testing. The principles presented in this thesis could be generalized for each multi-step architecture by taking into consideration its particularities.

Efficient reduced code linearity test techniques that provide full fault coverage will pave the way for the development of efficient ADC BIST technniques, since a major problem is the need to deal with all the codes of high resolution ADCs. There is no doubt that the holy graal of analog and mixed-signal test is a generalization of Built-In Self-Test techniques as is done in the world of digital circuits.

The servo-loop based BIST schemes can be very good candidates to be used along with a reduced code test scheme, as the codes to be measured can be tackled one after another. In [18], the authors have considered implementing the digital version of the servo-loop (Figure 7.4(a)) and they have used dynamic element matching in order to allow using low resolution and low accuracy DACs in the feedback loop. In our group we have started considering the oscillation BIST [16]. In order to test high resolution ADCs, using an integrator in the feedback loop will require a current source delivering a very small current and the integration time needs to be controlled very precisely. In order to circumvent these challenges, it was proposed to generate small voltage steps using a capacitor ratio. The required precision will be obtained since the capacitor matching is very good in new fabrication processes and is reported to be at least 1/10000.

Put aside the servo-loop based BIST schemes, the small triangular wave approach

[25, 26] might be a good candidate too. In a reduced code test scheme, the small triangular waves will not be offsetted by a constant amount, but the offset will vary depending on which code we intend to measure. Also, since the used amplitude is very small, the constraints on the linearity are relaxed. The on-chip generation methods of full scale triangular signals proposed in [27] and [28] could be easily adapted.

Also, the calibration schemes that have been proposed to compensate for the process-dependant variability of the slope of an integrated ramp [19, 21] could be used in the purpose of generating a ramp with varying slopes. The ramp will be slow at the positions of the codes we intend to measure and will be very fast at the positions of the codes we do not intend to measure. Furthermore, the Stimulus Error Identification and Removal techniques [29, 30] could be incorported to circumvent the inability to obtain a suitable linearity of the stimulus.

Chapter 7

Résumé en français

7.1 Introduction

7.1.1 Introduction

Le test de production est la dernière étape du flot de conception d'un circuit intégré. Selon le type du circuit la procédure de test de production est différente. Le test des circuits numériques (microprocesseurs, blocs logiques, mémoires...) consiste en l'injection de vecteurs de test qui permettent la détection de la présence de fautes. Les circuits défectueux sont repérés par une simple comparaison entre le résultat du test et le résultat idéal. La génération des vecteurs de test est faite durant la phase de conception du circuit. Le développement des outils de CAO a conduit à l'automatisation de ces procédures. Quant au test des circuits analogiques (amplificateurs opérationnelles, filtres, comparateurs, références de tension ou de courant) et mixtes (Convertisseurs Analogique-Numérique, Convertisseurs Numérique-Analogique, PLLs...), il est basé sur la vérification des spécifications du circuit. Son développement consiste en l'installation d'un environnement de test convenable (faible bruit, faible interférence, etc) et le choix des générateurs de signaux de test adéquats.

Étant donné que les performances et la complexité des produits sont en constante augmentation, le test des circuits analogiques et mixtes est en train de devenir de plus en plus problématique. Les Convertisseurs Analogique-Numérique (CAN) sont présents sur pratiquement tout système à signaux mixtes. Les CAN plus performants ont généralement une meilleure résolution. La Non-Linéarité Différentielle (NLD) et la Non-Linéarité Intégrale (NLI) sont les performances statiques les plus importantes mesurées lors d'un test de production. Dans un test de production standard, une sinusoïde ou une rampe saturées sont appliquées à l'entrée du CAN et le nombre d'occurrences de chaque code est obtenu, d'où l'on construit l'histogramme. A partir de cet histogramme la NLI et la NLD sont facilement calculées.

Cette approche requiert la collection d'une large quantité de données puisque chaque code doit être traversé plusieurs fois pour moyenner le bruit. La quantité de données nécessaire pour obtenir une bonne précision de mesure augmente exponentiellement avec la résolution du CAN sous test. Ainsi, le temps de test devient très long pour les CAN à haute résolution. Dans [8], il est cité que le test des circuits mixtes d'un SoC

moderne consume jusqu'à 30% du temps de test global, alors que les circuits mixtes occupent moins de 5% de la surface global.

7.1.2 Motivations et contributions

L'augmentation du temps de test se traduit par l'augmentation des coûts de test. A ce jour, le test statique des CANs à haute résolution est adressé avec les mêmes procédures utilisées pour les CAN à faible résolution et le temps de test devient excessif vis-à-vis de la surface de silicium testé ou du temps de test des autres blocs. Ceci implique un compromis temps de test/précision qui génère beaucoup d'instabilités et qui peut amener à jeter des circuits qui sont bons. Pour cette raison, la recherche de méthodes de test alternatives est entrain d'attirer de plus en plus d'attention.

Le projet de cette thèse a été conduit en collaboration avec STMicroeletronics, où le but était de trouver une solution pour diminuer le temps de test des CANs de type pipeline. Les CANs de type pipeline offrent un bon compromis entre la vitesse, la résolution et la consommation. C'est l'architecture la plus utilisée pour des résolutions supérieures à 8 bits et des fréquences d'échantillonnage d'une dizaine à des centaines de MHz.

Dans cette thèse l'architecture de ces CANs a été étudiée et des techniques efficaces ont été proposées pour l'application du test à code réduit. Le test à code réduit peut être appliqué aux CANs, qui, grâce à une particularité de leur architecture, ont des groupes de codes qui ont la même largeur. De ce fait, il est possible de ne considérer qu'un sous-groupe de codes lors du test, ce qui permet de réduire considérablement le temps de test.

7.2 État de l'art du test statique des CANs

7.2.1 Les principes du test statique des CANs

Définitions

La fonction de transfert idéale d'un CAN 3-bit est montrée sur la Figure 7.1. Lorsque le CAN est idéal tous les codes ont la même largeur, appelé LSB, qui est défini comme suit :

$$LSB = \frac{T_{N-1} - T_1}{N-2} \tag{7.1}$$

où N est le nombre de codes d'un CAN à n-bit ($2^n = N$) et T_i est la tension de transition du code i.

Les erreurs statiques les plus importantes sont la Non-Linéarité Différentielle (NLD) et la Non-Linéarité Intégrale (NLI) définies comme suit :

$$NLD(i) = \frac{T_{i+1} - T_i}{LSB} - 1, i \in [1, N-2] \tag{7.2}$$

$$NLI(i) = \sum_{k=1}^{i-1} NLD(k) \tag{7.3}$$

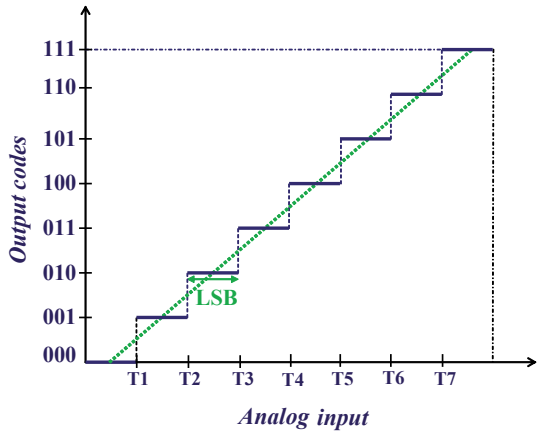


Figure 7.1: Fonction de transfert idéale d'un CAN.

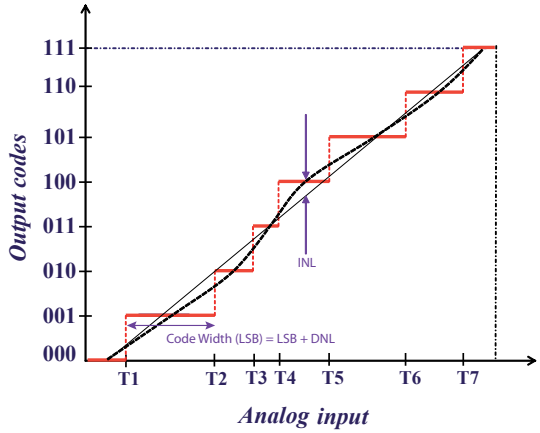


Figure 7.2: Illustration de la NLD et de la NLI sur une fonction de transfert d'un CAN non-idéale.

Comme illustré sur la Figure 7.2, la NLD d'un code reflète la déviation de sa largeur par rapport à la largeur idéale de 1 LSB. Quant à la NLI, elle reflète la déviation de la fonction de transfert par rapport à une fonction de transfert idéale, qui est parfaitement linéaire.

Le test statique

Le test statique des CANs consiste en l'extraction des performances statiques décrites ci-dessus. La méthode de l'histogramme et la méthode de la servo-loop sont les méthodes standards pour l'extraction de ces paramètres.

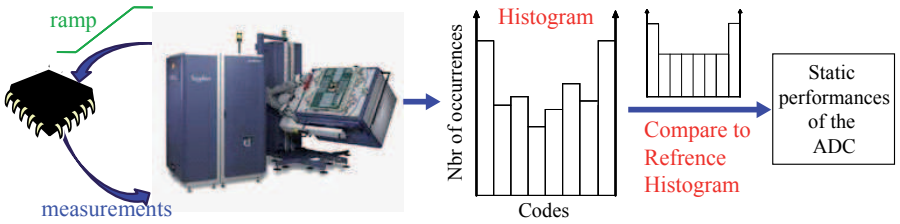


Figure 7.3: Le principe du test par histogramme.

a) La méthode d'histogramme

Dans la méthode d'histogramme, un signal analogique ayant une distribution connue est appliqué à l'entrée du CAN et le nombre d'occurrences de chacun des codes est obtenu. A partir de cela, les performances statiques du CAN peuvent être calculées en considérant l'histogramme de référence du signal utilisé en entrée. L'histogramme de référence pour une rampe linéaire en entrée est obtenu comme suit :

$$H_{ref}(0) = H_{ref}(2^n - 1) = N_T * \left(\frac{2^{n-1} * (A_{in} - FS) + FS}{2^n * A_{in}} \right) \tag{7.4}$$

$$H_{ref}(i) = N_T * \frac{FS}{2^n * A_{in}} \; ; i \in [1, 2^n - 2] \tag{7.5}$$

où $H_{ref}(i)$ est le nombre idéal d'occurrences du code i, N_T est le nombre total d'échantillons, n est la résolution du CAN, FS est la dynamique d'entrée du CAN et A_{in} est l'amplitude de la rampe. Si $H(i)$ est le nombre d'occurrences du code i, $NLD(i)$ est obtenue comme suit :

$$NLD(i) = \frac{H(i) - H_{ref}(i)}{H_{ref}(i)} LSB \tag{7.6}$$

Pour un signal sinusoïdal en entrée, tous les codes n'ont pas le même histogramme de référence. Dans le cas où le milieu du signal sinusoïdal correspond au milieu de la dynamique d'entrée du CAN, les histogrammes de référence sont donnés par :

$$H_{ref}(0) = H_{ref}(2^n - 1) = \frac{N_T}{\pi} * \left(arcsin \left[\left(\frac{1}{2^{n-1}} - 1 \right) * \frac{FS}{A_{in}} \right] - \frac{\pi}{2} \right) \tag{7.7}$$

$$H_{ref}(i) = \frac{N_T}{\pi} * \left(arcsin \left[\left(\frac{2 * i - 2^n}{2^n} \right) * \frac{FS}{A_{in}} \right] - arcsin \left[\left(\frac{2 * i - 2^n - 2}{2^n} \right) * \frac{FS}{A_{in}} \right] \right); \tag{7.8}$$

En pratique, le milieu du signal sinusoïdal correspond rarement au milieu de la dynamique du CAN. Dans ce cas, un offset V_o et l'amplitude vue par le CAN A_o doivent être inclus dans les calculs [10] :

$$V_o = \left(\frac{2^n}{2} - 1 \right) * \left(\frac{(cos(\pi * \frac{N2}{N})) - (cos(\pi * \frac{N1}{N}))}{(cos(\pi * \frac{N2}{N})) + (cos(\pi * \frac{N1}{N}))} \right) \tag{7.9}$$

$$A_o = \frac{(\frac{2^n}{2} - 1) - V_o}{cos(\pi * \frac{N1}{N})} \tag{7.10}$$

où $N1$ est le nombre d'occurrences du dernier code et $N2$ le nombre d'occurrences du premier code. Ainsi, l'histogramme de référence du code i est donné par :

$$H_{ref}(i) = \frac{N_T}{\pi} * \left(arcsin\left[\frac{i - (\frac{2^n}{2} - 1) - V_o}{A_o}\right] - arcsin\left[\frac{i - (\frac{2^n}{2}) - V_o}{A_o}\right]\right); i \in [1, 2^n - 2] \tag{7.11}$$

b) La méthode de Servo Loop

Le principe du test par servo-loop est illustré sur la Figure 7.4. Dans cette approche, une tension analogique est appliquée à l'entrée du CAN et le code numérique de sortie est comparé à un code de référence k. La boucle asservie permet de diminuer ou augmenter la tension d'entrée jusqu'à atteindre la tension de transition du code k, $T(k)$ [9],[13], [14] and [15].

7.2.2 État de l'art des méthodes de test alternatives

Les différents travaux visent la réduction du temps et/ou coût du test. On peut les classifier en deux groupes : techniques de conception en vue du test (DFT) et techniques de test intégré (BIST).

(a) Techniques de test intégré (BIST)

BIST par oscillation

La Figure 7.5 montre le principe de cette technique. Elle est inspirée de la technique de test par servo-loop, où dans le cas intégré le système est forcé d'osciller entre les transitions de deux codes pré-déterminés. Ainsi, la fréquence d'oscillation dépendra du temps de conversion et des tensions de transitions des codes pré-déterminés.

Mesure de NLD et NLI basée sur un comptage

Le principe de ces techniques est basé sur le fait que le bit le moins significatif (LSB) d'un CAN fait toujours une transition d'un code au code suivant. De ce fait, la largeur d'un code peut être déduite en calculant le nombre d'incréments d'un compteur qui séparent deux transitions successives du LSB. L'inconvénient de ce type de techniques est leur sensibilité au bruit.

BIST statique numérique

Dans [18], un générateur de stimulus qui permet de mesurer les performances statiques d'un code prédéterminé a été proposé. La méthode est inspirée de la méthode de test par Servo-Loop (Figure 7.6) et elle est compatible pour être utilisée dans un environnement de test numérique.

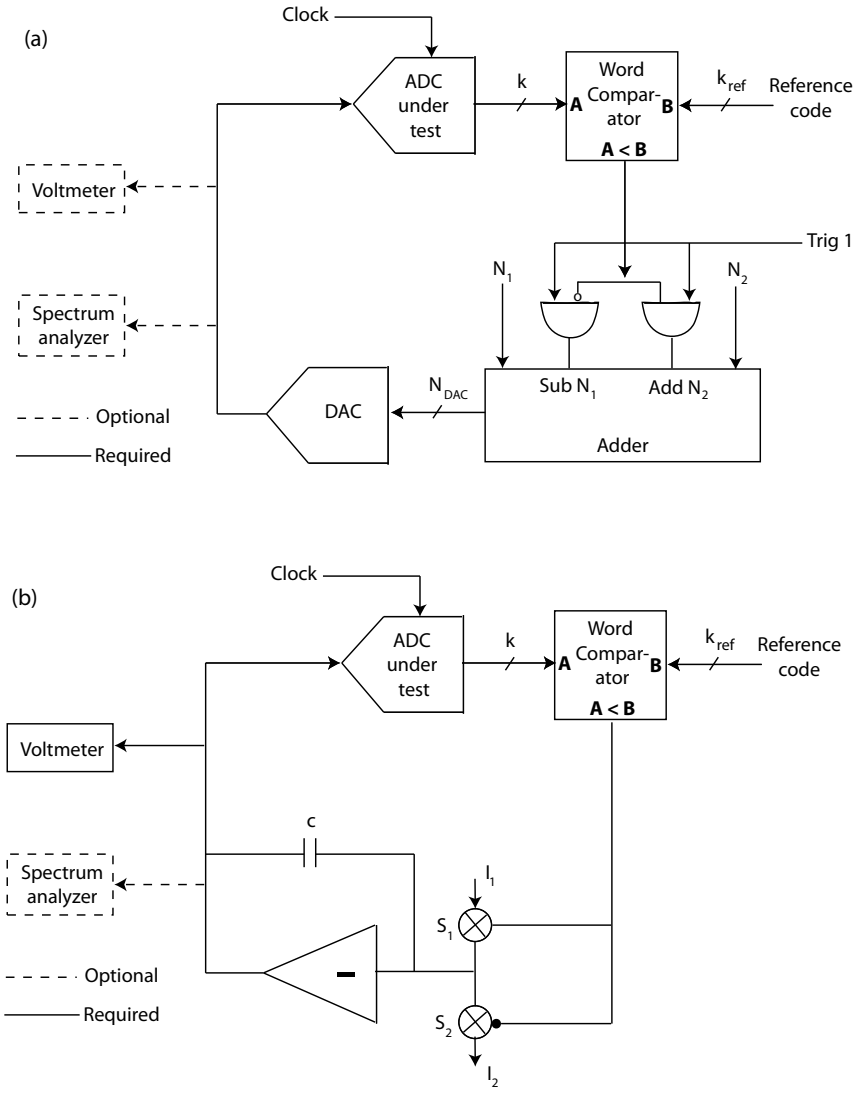


Figure 7.4: La servo-loop : (a) approche numérique, et (b) approche analogique.

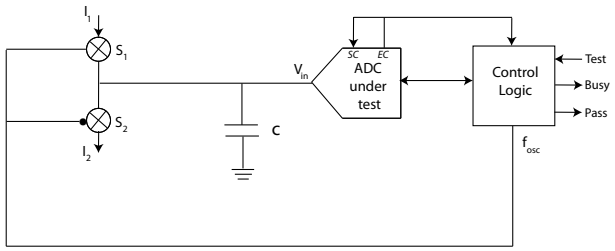


Figure 7.5: BIST par oscillation.

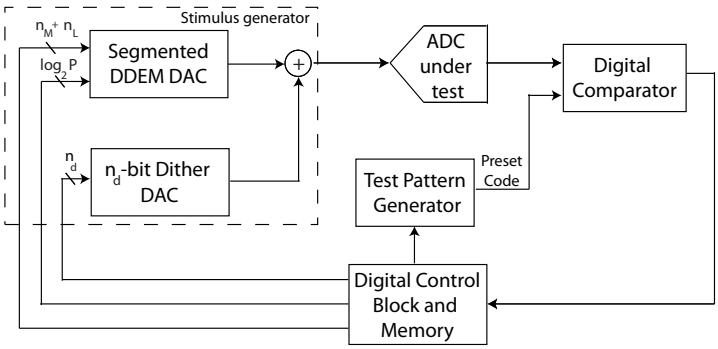


Figure 7.6: BIST statique numérique.

Générateur de rampe intégré pour le test statique

La génération d'une rampe pour le test intégré a été considérée dans [19], [20],[21],[22],[23] et [24]. Le principe de base est de charger une capacité par un courant constant. La meilleure linéarité reportée obtenue par cette méthode est limitée à 10 à 11 bits, ce qui est au-dessous des besoins de test statique pour des CANs de haute résolution.

b) Techniques de conception en vue du test (DFT)

Test par histogramme utilisant des signaux triangulaires à faible amplitude

Dans [25] et [26], il a été proposé d'utiliser des signaux triangulaires de faible amplitude afin de construire un histogramme. Ce test permet d'alléger les besoins de linéarité sur le stimulus du test puisque l'amplitude des signaux triangulaires est beaucoup plus faible que la dynamique d'entrée du CAN. La procédure de cette technique est décrite dans la Figure 7.7.

Identification et suppression de la non-linéarité du stimulus

Les méthodes classiques du test par histogramme requièrent que la linéarité du stimulus soit au moins 2 bit plus élevée que la linéarité du CAN sous test. Ce qui est d'autant plus difficile à satisfaire pour les CANs à haute résolution. Dans [29] et [30], une technique qui permet d'utiliser un stimulus ayant une linéarité plus faible que la linéarité du CAN sous test a été démontrée.

Test de NLD et NLI basé sur un modèle

Le nombre d'échantillons qui doit être acquis dans la technique standard de test statique est d'autant plus grand que la résolution du CAN est élevée et que le niveau de bruit environnant est haut. Les techniques de test se basant sur un modèle ont été introduites comme un moyen qui permet, en utilisant un modèle, de réduire le temps de test statique d'un CAN en réduisant le nombre d'échantillons qui doit être acquis [5], [32] et [6] .

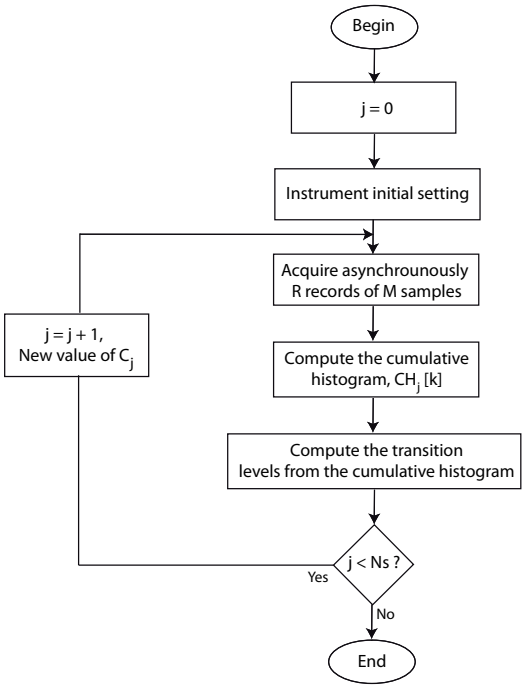


Figure 7.7: La procédure de test par histogramme utilisant des signaux triangulaires.

Le test de linéarité à code réduit

Certaines architectures de CAN offrent la possibilité d'appliquer le test à code réduit. La réduction du temps de test statique est possible car nous pouvons compter sur la mesure d'un ensemble réduit de largeurs de code judicieusement sélectionnées pour extraire la caractéristique statique complète du CAN. Les travaux publiés à ce jour ont concerné les CAN de type SAR [33], [34] et pipeline [7], [35], [36].

Les CANs de type SAR ont relativement des faibles résolutions et sont plus lents, comparés aux CANs de type pipeline, qui sont plus rapides et ont des résolutions plus élevées. Le test à code réduit est intéressant dans les deux cas puisque le temps de test est d'autant plus grand que la fréquence d'échantillonnage est faible et que le nombre de codes du CAN est plus élevé.

Dans ce travail, nous avons développé une technique de test à code réduit pour les CANs de type pipeline. Nous montrerons les limitations de la technique existantes et nous expliquerons en détail la technique proposée, qui permet d'appliquer le test à code réduit d'une manière très efficace.

7.3 Test de linéarité à code réduit pour les CANs de type pipeline

7.3.1 Les CANs de type pipeline

Un CAN de type pipeline se compose d'une cascade d'étages comme représenté sur la Figure 7.8. Chaque étage est constitué d'un échantillonneur-bloqueur (S/H), un sous-CAN, un sous-CNA, un soustracteur et un amplificateur. Le signal d'entrée de l'étage est tout d'abord converti par le sous-CAN à un code numérique qui est la sortie de l'étage. Le résultat de la conversion est reconvertie par le sous-CNA en un signal analogique et ensuite soustrait du signal d'entrée. Le résultat de la soustraction est amplifié de façon à utiliser la même tension de référence dans tout les étages. Le résidu du premier étage est échantillonné par le deuxième étage et ainsi de suite. La logique numérique rassemble les sorties numériques de tout les étages et fournit la sortie du CAN [37, 38, 39].

7.3.2 Le principe du test à code réduit pour les CANs de type pipeline

Les résidus d'un étage pipeline

La Figure 7.9 montre les résidus du premier et deuxième étages 1.5-bit d'un CAN pipeline. Le nombre placé au-dessus d'une transition indique lequel des deux comparateurs de l'étage est exercé (son seuil est franchi) à cette transition .

Comme on le voit, si nous parcourons la dynamique d'entrée du CAN, les deux comparateurs du premier étape sont exercés une seule fois. Les trois segments du résidu du premier étage traversent les intervals $[-V_{ref}, V_{ref}/2]$, $[-V_{ref}/2, V_{ref}/2]$, et $[V_{-ref}/2, V_{ref}]$, respectivement. Ainsi, pour chaque segment du résidu du premier étage, chacun des deux comparateurs du deuxième étage est exercé une fois. Par conséquent, si nous parcourons la dynamique d'entrée du CAN, les deux comparateurs du deuxième étage sont exercés trois fois chacun au total. De façon similaire, si nous parcourons la dynamique d'entrée du CAN, les deux comparateurs du troisième étage seront exercés sept fois chacun. Ceci peut être vu sur la Figure 7.10 où le résidu du troisième étage est inclu aussi.

Le principe

A partir du deuxième étage d'un CAN de type pipeline, les comparateurs de chaque étage sont exercés plusieurs fois. Ceci implique que, à la sortie numérique du CAN, il y aura des transitions qui sont dues au même comparateur, comme le montre la Figure 7.11.

Les transitions de sortie du CAN qui impliquent le même comparateur sont toutes affectées de la même manière en présence d'erreurs dues à des variations process. Ainsi, pour extraire la caractéristique de transfert complète, il suffit de considérer seulement un ensemble représentatif de transitions de sortie qui couvre tous les comparateurs dans tous les étages.

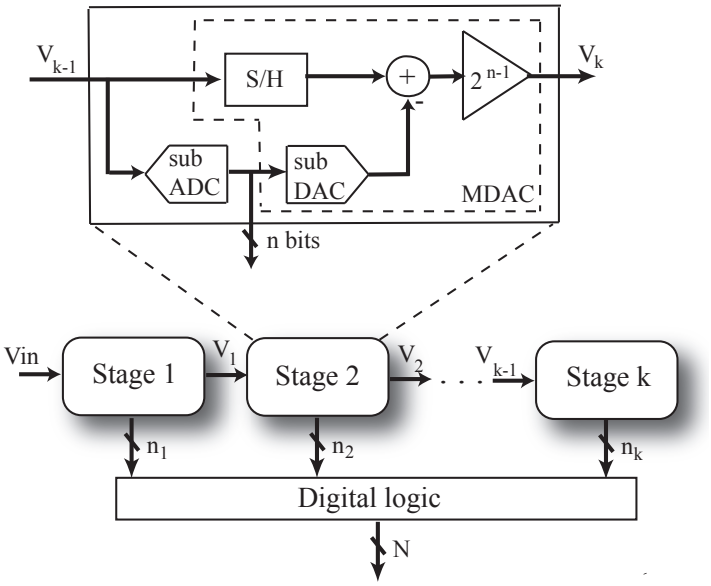


Figure 7.8: Architecture d'un CAN de type pipeline.

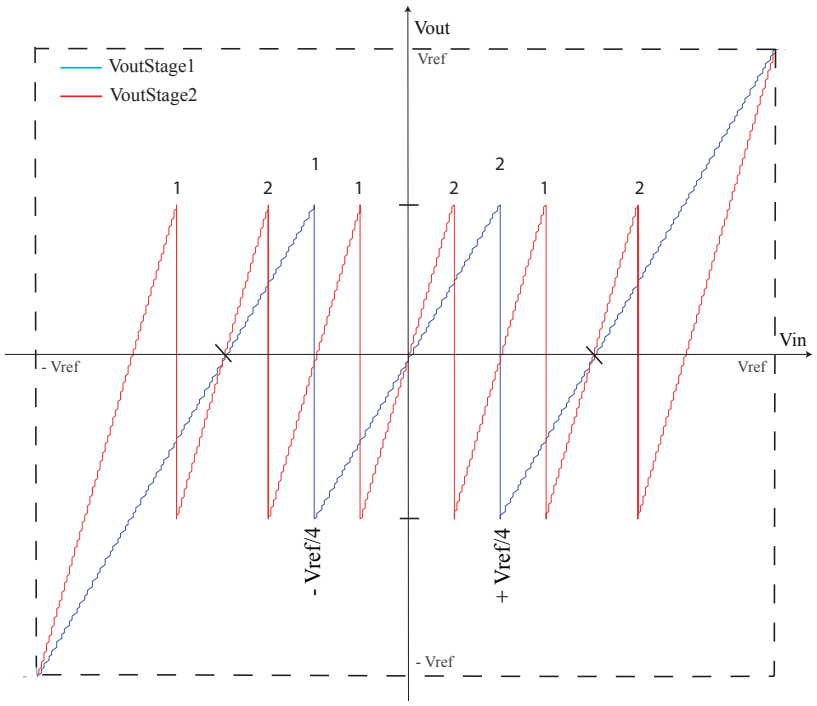


Figure 7.9: Résidu du premier et du deuxième étages d'un CAN pipeline à 1.5-bit/étage.

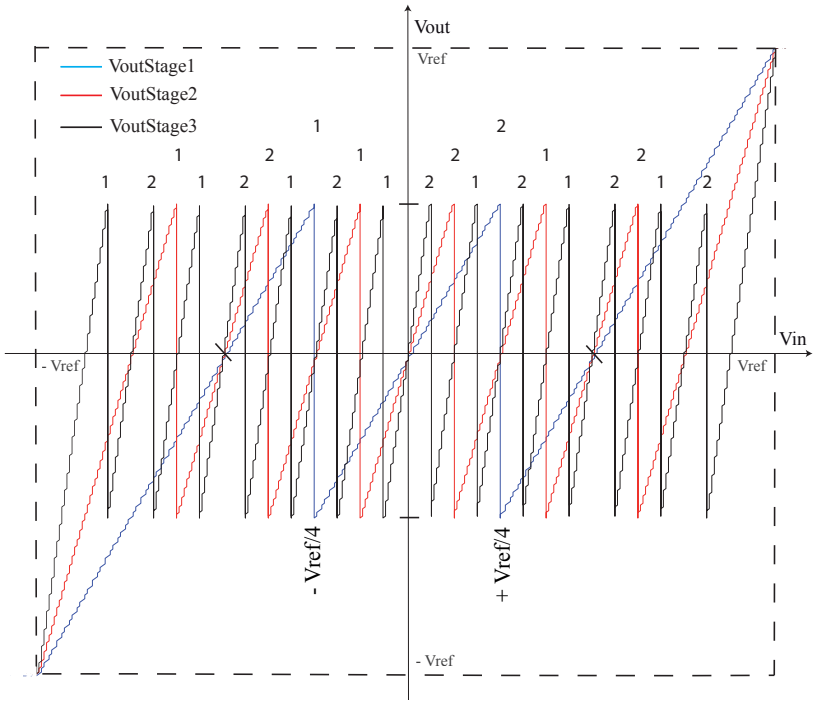


Figure 7.10: Résidu des trois premiers étages d'un CAN pipeline à 1.5-bit/étage.

Chaque comparateur dans chaque étage doit être représenté une fois dans cet ensemble. Pour l'ensemble des transitions de sortie choisies, nous mesurons les largeurs des codes qui les entourent. Par la suite, on peut facilement utiliser ces mesures pour en déduire les largeurs des codes autour des transitions qui n'ont pas été sélectionnées. Dans l'exemple de la Figure 7.11, les largeurs des codes λ et $\lambda + 1$ sont égales aux largeurs des codes μ et $\mu + 1$, respectivement, puisque ces deux transitions sont dues au même comparateur.

Ainsi, nous devons mesurer la largeur soit de λ ou de μ et la largeur soit de $\lambda + 1$ ou de $mu + 1$. De cette façon nous remplissons l'histogramme en s'appuyant sur un nombre réduit de mesures de largeur de code, ce qui se traduit par une réduction importante du temps de test statique.

7.3.3 Transitions naturelles et transitions forcées

Pour expliquer ces deux types de transitions, prenons les résidus idéaux des deux premiers étages 1.5-bit d'un CAN pipeline, comme le montre la Figure 7.12(a). La Figure montre également la sortie du sous-CNA de ces deux premiers étages.

En regardant le résidu du deuxième étage, on peut observer les six transitions correspondantes aux deux comparateurs. Le premier comparateur est exercé à trois reprises (transitions $00 \rightarrow 01$). Le deuxième comparateur est exercé également à trois reprises (transitions $01 \rightarrow 10$). Même s'il n'y a que six transitions de comparateurs dans la sortie du deuxième étage, nous observons que la sortie numérique, en plus des

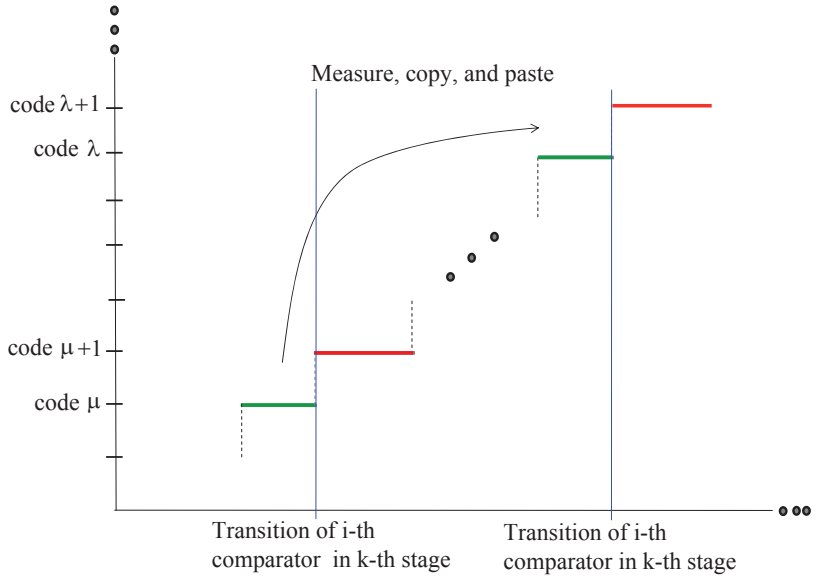


Figure 7.11: Principe du test à code réduit des CANs de type pipeline.

transitions 00 → 01 et 01 → 10, contient deux autres transitions (transitions 10 → 00). Ces deux transitions de la sortie numérique du deuxième étage ne correspondent pas à une transition dans le résidu du deuxième étage, mais elles se produisent sous l'influence de transitions dans le résidu du premier étage (voir la Figure 7.12(a)). Le résidu du premier étage devient soudainement inférieur aux seuils des comparateurs du deuxième étage ce qui cause une transition de la sortie numérique de cet étage de 10 à 00.

Nous appelons ces transitions 'transitions forcées' parce que la sortie numérique de l'étage change, mais ce changement n'est pas dû au fait que le seuil de l'un des comparateurs de l'étage a été franchi. Inversement, lorsque la sortie numérique de l'étage change à cause du seuil de l'un des comparateurs de l'étage étant franchi, la transition est appelée 'transition naturelle'

7.3.4 La technique

Pour que la NLD et la NLI estimées soient correctes, deux conditions principales doivent être atteintes. Tout d'abord, nous devons nous assurer que la transition de sortie du CAN est associée au bon comparateur. Ensuite, afin de sélectionner une transition de sortie du CAN qui est représentative d'un comparateur dans un étage donné il faut éviter de choisir une transition de sortie du CAN qui implique, en plus de ce comparateur, un comparateur de l'un des étages précédents.

Pour satisfaire ces conditions, nous avons proposé de surveiller les changements de la sortie numérique de chacun des étages avant qu'elle ne subisse la correction numérique, comme le montre la Figure 7.13. La raison en est que, lorsqu'un seuil d'un comparateur est franchi, cela produit nécessairement une transition dans la sortie numérique de l'

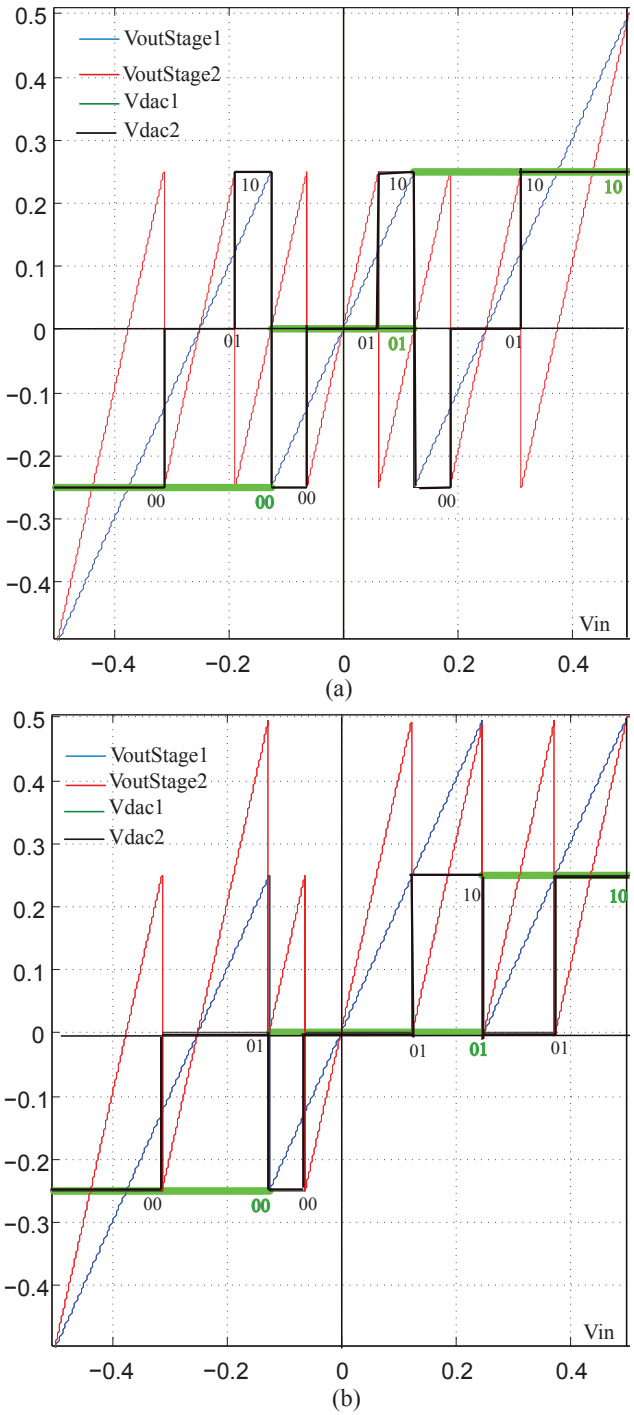


Figure 7.12: Résidu des deux premiers étages d'un CAN pipeline à 1.5-bit/étage superposés sur la sortie du sous-CNA : (a) nominal; (b) en présence d'offset de comparateur.

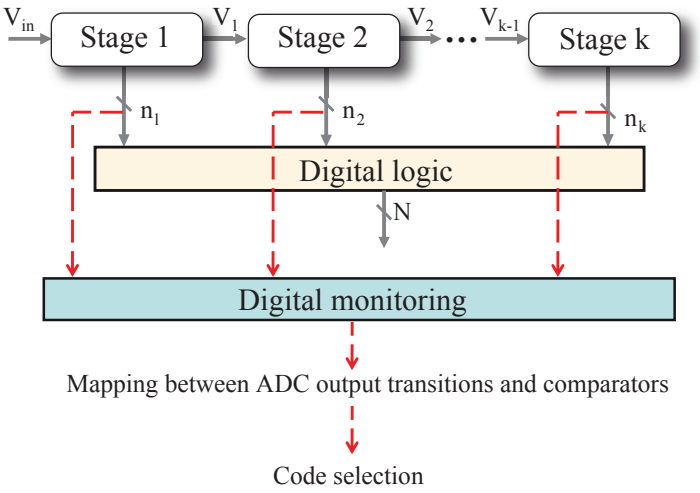


Figure 7.13: Surveillance des sorties numériques pour une correspondance correct entre les transitions de sortie du CAN et les comparateurs exercés.

étage auquel ce comparateur appartient. Une transition d'un certain code à un autre dans la sortie numérique d'un étage fournit une information complète par rapport à quel comparateur a été exercé. En effet, une transition à la sortie numérique d'un étage peut être mise en correspondance avec la transition de sortie du CAN résultante en traitant simplement les sorties des différents étages comme cela est fait par le bloc logique numérique du CAN.

Si l'on considère une rampe ascendante, une 'transition naturelle' correspond à une augmentation d'un pas de la sortie numérique de l'étage. Si la sortie numérique de l'étage diminue ou augmente par plus d'un pas, la transition est une 'transition forcée'. Comme expliqué ci-dessus, une transition forcée à la sortie d'un étage est due à une transition naturelle qui s'est produite dans un étage précédent.

Le tableau 7.1 répertorie toutes les transitions possibles qui peuvent se produire à la sortie numérique d'un étage 2.5-bit lors de l'opération du CAN en supposant une rampe ascendante. La sortie numérique de l'étage à un certain instant est notée par X, alors que la sortie numérique de l'étage au cycle d'horloge suivant est notée par Y. Il existe six transitions naturelles possibles correspondantes aux six comparateurs de l'étage, notées par $C1$ à $C6$. Fxy indique une transition forcée du code x au code y. Les cases grisées indiquent que la sortie numérique n'a pas changé. Dans l'ensemble, la surveillance numérique doit indiquer lequel des 42 scénarios du tableau a eu lieu pour chacun des étages.

Pour résumer, les étapes de la technique sont comme suit :

1. Trouver la correspondance entre les transitions de sortie du CAN et les comparateurs qui sont exercés.

2. Sélectionner un ensemble représentatif de transitions de sortie du CAN .

Table 7.1: Les différentes transitions possibles d'un étage 2.5-bit.

Y \ X	000	001	010	011	100	101	110
000		F10	F20	F30	F40	F50	F60
001	C1		F21	F31	F41	F51	F61
010	F02	C2		F32	F42	F52	F62
011	F03	F13	C3		F43	F53	F63
100	F04	F14	F24	C4		F54	F64
101	F05	F15	F25	F35	C5		F65
110	F06	F16	F26	F36	F46	C6	

3. Mesurer les largeurs des codes autour des transitions de sortie du CAN qui étaient sélectionnées .

4. Déduire les largeurs des codes autour des transitions de sortie du CAN qui n'ont pas été sélectionnées dans l'étape 2.

5. Calculer la NLD et la NLI à partir des largeurs de code obtenues .

7.3.5 Résultats de simulation

Les Figures 7.14(a) et 7.15(a) montrent la NLD et la NLI obtenues en utilisant la technique de l'histogramme standard.

La NLD et la NLI estimées en utilisant la technique de test à code réduit sont montrées sur les Figures 7.14(b) et 7.15(b). Comme on peut l'observer, les erreurs de linéarité maximales ont été correctement capturées et le maximum et minimum de NLD et NLI ont été estimées correctement.

7.4 Méthode pour l'élimination de l'effet du bruit

L'efficacité de la technique de test à code réduit a été démontrée dans le chapitre précédent en utilisant un modèle comportemental. Dans ce chapitre, nous allons montrer les problèmes qui peuvent être rencontrés lors de l'expérimentation dans un environnement bruyant. Ensuite, nous allons montrer une autre propriété importante des CANs de type pipeline et nous présenterons une méthode qui exploite cette propriété et qui permet de contourner les erreurs d'estimation qui peuvent survenir en raison de la présence du bruit.

7.4.1 Transitions bruyantes

L'étape la plus importante de la technique est de réaliser une correspondance correcte entre les transitions de sortie du CAN et les comparateurs exercés. Une fois qu'une correspondance est obtenue, il est possible de grouper les codes qui ont la même largeur.

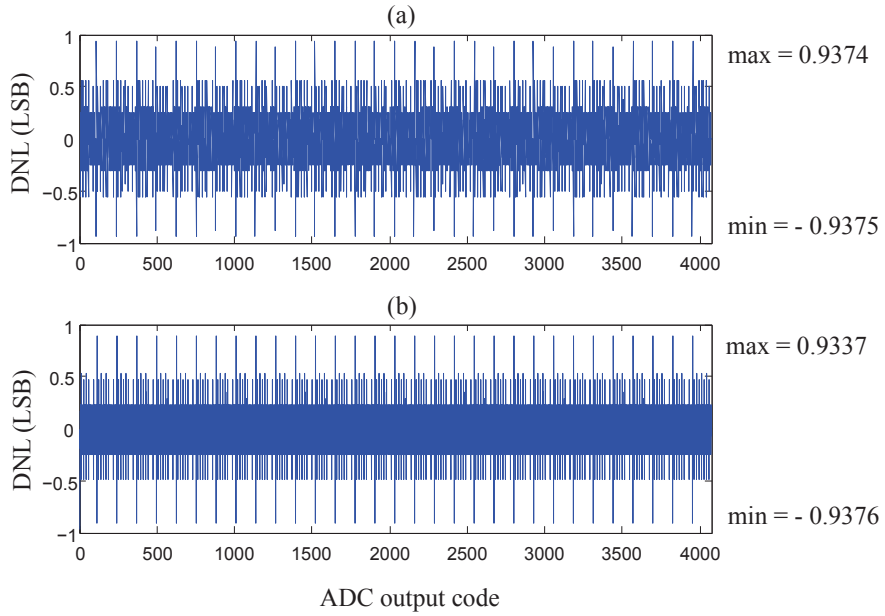


Figure 7.14: NLD d'un CAN pipeline 12-bit à 2.5-bit/étage : (a) technique standard par histogramme, et (b) la techique proposée.

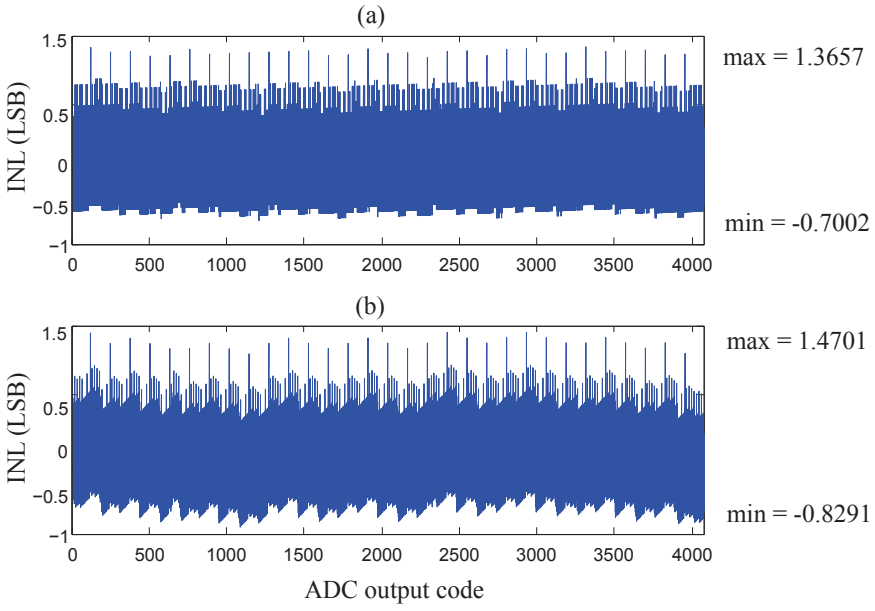
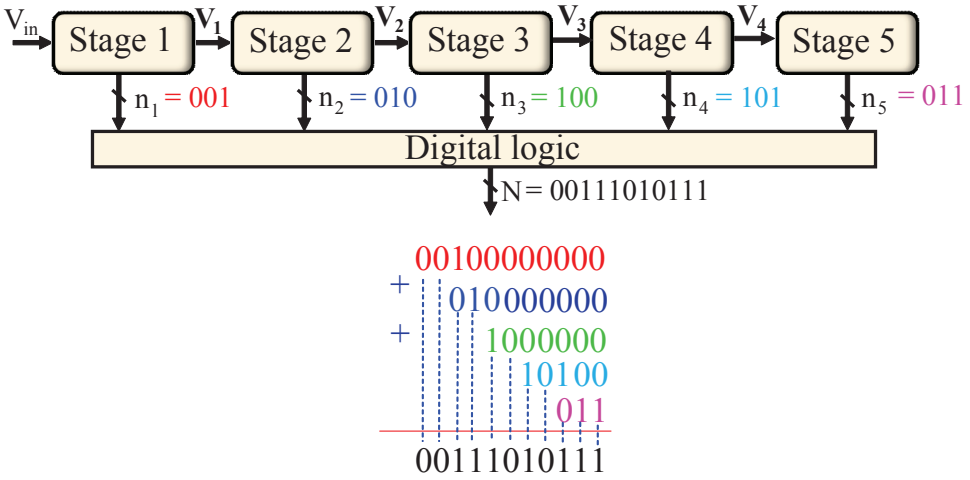


Figure 7.15: NLI d'un CAN pipeline 12-bit à 2.5-bit/étage : (a) technique standard par histogramme, et (b) la techique proposée.



$$N_{dec} = n_{1,dec} *2^{w1} + n_{2,dec} *2^{w2} + n_{3,dec} *2^{w3} + n_{4,dec} *2^{w4} + n_{5,dec} *2^{w5}$$
$$= 1 *2^{8} + 2 *2^{6} + 4 *2^{4} + 5 *2^{2} + 3 *2^{0}$$
$$= 471$$

Figure 7.16: La sortie du CAN en fonction de la contribution et du poids de chacun des étages.

Dans la Figure 7.16 nous montrons un CAN pipeline à 5 étages où chaque étage fournit un code numérique de 3 bits. Le code de sortie du CAN est calculé en faisant un chevauchement de 1 bit entre les sorties des étages. La sortie numérique du CAN peut être exprimée en décimal en fonction des sorties des étages individuels comme suit :

$$N_{dec} = n_{1,dec} * 2^{w1} + n_{2,dec} * 2^{w2} + n_{3,dec} * 2^{w3} + n_{4,dec} * 2^{w4} + n_{5,dec} * 2^{w5} \quad (7.12)$$

Dans ce qui suit nous allons nous référer à w_k comme étant le poids de l'étage k et au coefficient $n_{k,dec}$ comme étant la contribution de l'étage k. La Figure 7.16 montre un exemple de calcul du code de sortie du CAN correspondant à l'exemple donné.

Dans la Figure 7.17 nous superposons les sorties numériques des étages du CAN expérimental. La sortie de chacun des étages (en décimal) est montrée avec une couleur différente. Nous pouvons voir que certaines transitions sont très bruyantes et cela conduira à des imprécisions lors du calcul des correspondances entre la sortie du CAN et les transitions des comparateurs. Nous montrons cela avec deux exemples sur la Figure 7.17. Dans le premier exemple, l'intention était de calculer le code de sortie du CAN à droite de la transition du troisième comparateur du quatrième étage. On peut voir que :

- la sortie du premier étage à droite de cette transition est '4' ,
- la sortie du deuxième étage à droite de cette transition est '2' ,

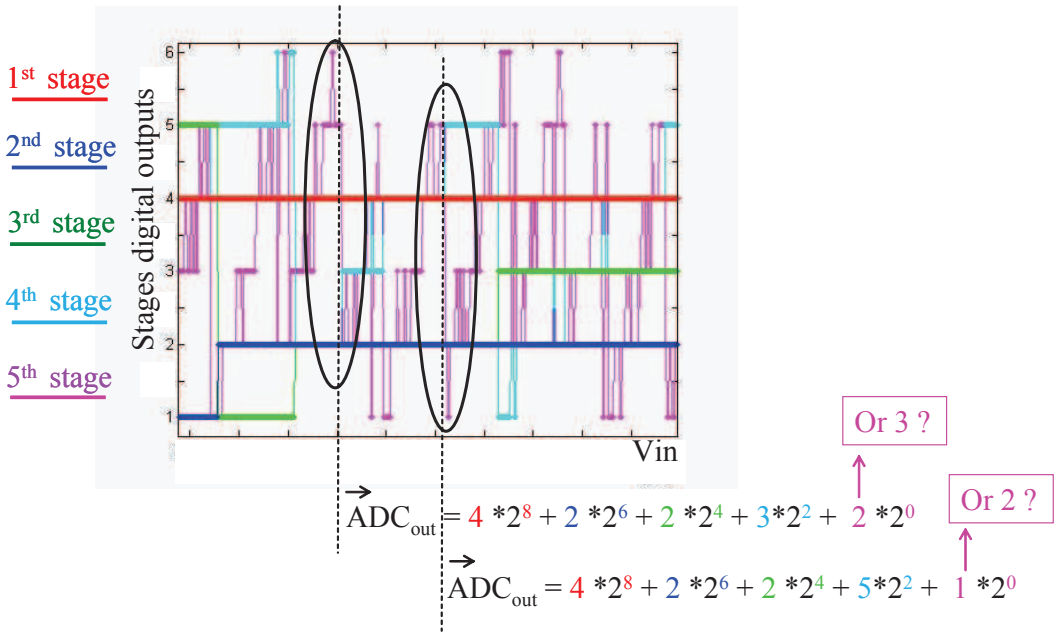


Figure 7.17: L'effet du bruit sur les transitions.

- la sortie du troisième étage à droite de cette transition est '2' ,
- la sortie du quatrième étage à droite de cette transition est '3' .

Quant au cinquième étage, si nous lisons la valeur de sortie qui est immédiatement à droite de la transition, nous trouvons une valeur de '2'. Visuellement, cela aurait pu être une valeur de '3'. En raison de la présence du bruit, la sortie numérique du cinquième étage à droite de la transition en question alterne entre '2' et '3'. Pour le deuxième exemple, si nous lisons la valeur de sortie qui est immédiatement à droite de la transition nous trouvons une valeur de '1'. Mais visuellement il semble que la valeur de '1' est un 'glitch' et que la valeur qui doit être prise est '2'.

Ainsi, en présence du bruit, en considérant les sorties des étages immédiatement à gauche ou à droite d'une transition on peut avoir une correspondance erronée entre les sorties du CAN et les comparateurs exercés. Cela se traduira par un mauvais groupement des codes de sortie du CAN, ce qui résultera en une estimation erronée de NLD et NLI.

7.4.2 Les codes racines

Pour les explications de cette section, nous utilisons un modèle comportemental sans bruit d'un CAN 10 bits qui comprend quatre étages 2.5-bits et un dernier étage 2 bits. Nous allons considérer le troisième comparateur du deuxième étage de ce CAN .

La Figure 7.18 superpose la sortie du CAN sur la sortie du deuxième étage. depuis cette figure on peut identifier les codes de sortie du CAN qui sont associés à chacune

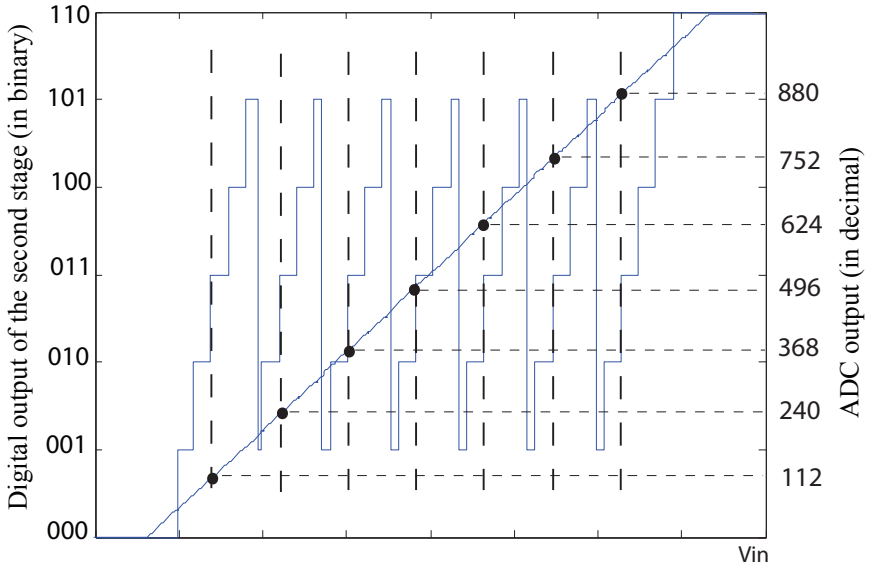


Figure 7.18: Transitions dans le deuxième étage et les sorties correspondantes du CAN.

des transitions du troisième comparateur du deuxième étage.

Pour ce CAN le poids du premier étage est égal à 2^7. Nous pouvons écrire les codes de sortie du CAN représentés sur la Figure 7.18 sous la forme :

$$
\begin{aligned}
112 &= 112 + 0 \cdot 2^7, \\
240 &= 112 + 1 \cdot 2^7, \\
368 &= 112 + 2 \cdot 2^7, \\
496 &= 112 + 3 \cdot 2^7, \\
624 &= 112 + 4 \cdot 2^7, \\
752 &= 112 + 5 \cdot 2^7, \\
880 &= 112 + 6 \cdot 2^7.
\end{aligned}
$$

A noter que le coefficient qui est multiplié par le poids du premier étage dans les équations ci-dessus correspond à la valeur de la sortie du premier étage dans la zone où le code de sortie du CAN est pris. Tous les codes de sortie du CAN représentés sur la Figure 7.18 peuvent être obtenus à partir du code '112' par addition d'un terme qui est obtenu en multipliant la valeur de la sortie du premier étage par le poids du premier étage.

Nous référons au code '112' comme étant le *code racine droit* du troisième comparateur du deuxième étage. De même, en regardant les codes de sortie du CAN à gauche de la transition du troisième comparateur du deuxième étage, on peut définir le *code racine gauche* du troisième comparateur du deuxième étage.

Pour généraliser, divisons les étages du CAN en deux groupes. Le premier groupe contient les étages 1 à $k-1$ et le second groupe contient les étages k à N, où N est le nombre total d'étages. Nous définissons également une fonction $f(x, comp_k^i, w)$ où :

- x est le numéro d'étage du CAN,
- $comp_k^i$ est le $i^{ième}$ comparateur du $k^{ième}$ étage,
- w réfère au côté droit R ou au côté gauche L de la transition du comparateur $comp_k^i$.

On définit $f(x, comp_k^i, w)$ comme suit :

> "Compte tenu de la transition du $i^{ième}$ comparateur du $k^{ième}$ étage, $f(x, comp_k^i, w)$ est la valeur de la sortie numérique du $x^{ième}$ étage sur le côté w de cette transition."

Chaque fois que le $i^{ième}$ comparateur est exercé dans le $k^{ième}$ étage, la sortie numérique du $k^{ième}$ étage transite d'une valeur égale à $f(k, comp_k^i, L)$ à une valeur égale à $f(k, comp_k^i, R)$. A chaque fois que le même comparateur est exercé dans un étage, le résidu de cet étage, qui est l'entrée analogique pour les étages suivants transite toujours entre les deux mêmes valeurs analogiques. Cela implique que chaque fois que le $i^{ième}$ comparateur est exercé dans le $k^{ième}$ étage, la sortie numérique du $x^{ième}$ étage ($x = k+1, \cdots, N$), est toujours égale à la valeur $f(x, comp_k^i, L)$ avant la transition et à la valeur $f(x, comp_k^i, R)$ après la transition.

On définit :

$$L_i^k = [f(k, comp_k^i, L), f(k+1, comp_k^i, L), \cdots, f(N, comp_k^i, L)] \tag{7.13}$$

$$R_i^k = [f(k, comp_k^i, R), f(k+1, comp_k^i, R), \cdots, f(N, comp_k^i, R)] \tag{7.14}$$

Si nous additionnons les éléments de L_i^k en tenant en compte le poids de chacun des étages, on obtient alors le code racine gauche du $i^{ième}$ comparateur du $k^{ième}$ étage. Si nous additionnons les éléments de R_i^k en tenant en compte le poids de chacun des étages, on obtient alors le code racine droit du $i^{ième}$ comparateur du $k^{ième}$ étage.

7.4.3 Elimination de l'effet du bruit

Nous avons montré ci-dessus que les codes de sortie du CAN correspondants au même comparateur peuvent être exprimés comme une fonction des codes racine de ce comparateur et de la contribution des étages précédents l'étage où appartient ce comparateur. Cette propriété peut être utilisée pour éliminer l'effet de la présence du bruit lors des calculs des correspondances entre les transitions à la sortie du CAN et les transitions des comparateurs.

Nous appliquons d'abord une rampe et nous observons la nature des transitions dans chacun des étages. Pour chaque transition naturelle du $i^{ième}$ comparateur dans le $k^{ième}$ étage nous obtenons L_i^k et R_i^k. Si n est le nombre de transitions naturelles, alors nous avons n valeurs de chaque élément $f(x, comp_k^i, L)$ de L_i^k et n valeurs de chaque élément $f(x, comp_k^i, R)$ de R_i^k, $x = k, \cdots, N$.

En raison de la présence du bruit, les valeurs extraites des éléments de L_i^k et R_i^k pour $x \geq k+1$ ne sont pas nécessairement les mêmes pour chaque transition naturelle

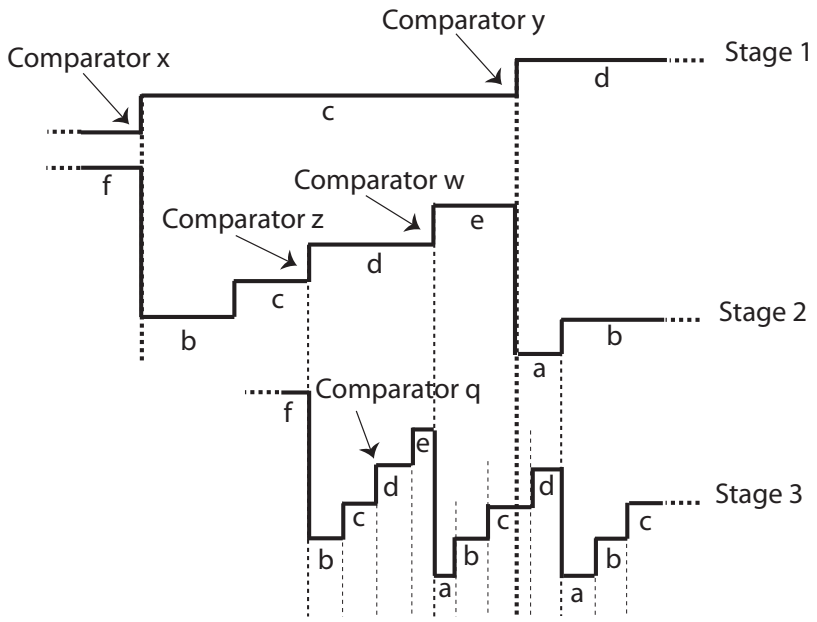


Figure 7.19: Aperçu de transitions de trois premiers étages pipeline.

du même comparateur. En d'autres termes, les codes racine gauche et droit calculées à partir de différentes transitions naturelles peuvent ne pas être les mêmes.

Pour $x = k+1, \cdots, N$, on considère $\mu^{x,L}_{comp^i_k}$ et $\mu^{x,R}_{comp^i_k}$ les valeurs $f(x, comp^i_k, L)$ et $f(x, comp^i_k, R)$, respectivement, qui apparaissent le plus fréquemment parmi les n valeurs extraites. De cette façon on obtient les L^k_i et R^k_i non-bruités :

$$L^k_i = [i-1, \mu^{k+1,L}_{comp^i_k}, \cdots, \mu^{N,L}_{comp^i_k}], \tag{7.15}$$

$$R^k_i = [i, \mu^{k+1,R}_{comp^i_k}, \cdots, \mu^{N,R}_{comp^i_k}]. \tag{7.16}$$

A partir des L^k_i et R^k_i non-bruités on peut calculer les codes racine droit et gauche non-bruités à partir desquelles une correspondance très précise est obtenue entre les transitions des comparateurs exercés et les transitions de sortie du CAN.

7.4.4 Principe du calcul des correspondances en utilisant les codes racine non-bruités

La Figure 7.19 montre un aperçu des transitions de trois étages consécutives, elle sera utilisée pour visualiser les explications qui vont suivre.

Soit lrc^i_k et rrc^i_k les codes racine gauche et droit, respectivement, du $i^{ième}$ comparateur dans le $k^{ième}$ étage. w_k représentera le poids du $k^{ième}$ étage. Dans le premier étage, les codes racine gauche et droit des comparateurs sont eux-mêmes les codes de

sortie du CAN. Par exemple, en se référant à la Figure 7.19, les codes de sortie lrc_1^x et rrc_1^x correspondent au comparateur x du premier étage.

Pour compléter les calculs, nous devons trouver pour un comparateur dans le $k^{ième}$ étage, $k \geq 2$, les combinaisons des sorties des étages 1 à $k-1$ qui doivent être ajoutés aux codes racines pour calculer les codes de sortie du CAN correspondants.

Considérons deux transitions consécutives du $k^{ième}$ étage, où $k \geq 1$. Cette paire de transitions peut être (naturelle, naturelle), (naturelle, forcée), ou (forcée, naturelle). Entre ces deux transitions la sortie des étages x ($x = 1, \cdots, k$), est fixée à une valeur Y_x pendant que la sortie numérique du $(k+1)^{ième}$ étage rampe de Y_{k+1} à $Y_{k+1} + t$, où t est le nombre de comparateurs qui ont été exercés. Par exemple, sur la Figure 7.19, entre deux transitions naturelles consécutives du premier étage qui sont dues aux comparateurs x et y, la sortie numérique du premier étage est constante à une valeur égale à c, alors que la sortie numérique du deuxième étage rampe de b à e, et $t = 3$.

Si le $i^{ième}$ comparateur dans le $(k+1)^{ième}$ étage est exercé lorsque sa sortie change de Y_{k+1} à $Y_{k+1}+t$, alors les deux codes de sortie du CAN $[lrc^i_{(k+1)} + Y_k \cdot 2^{w_k} + \cdots + Y_1 \cdot 2^{w_1}]$ et $[rrc^i_{(k+1)} + Y_k \cdot 2^{w_k} + \cdots + Y_1 \cdot 2^{w_1}]$ correspondent au $i^{ième}$ comparateur dans le $(k+1)^{ième}$ étage.

Par exemple, sur la Figure 7.19, les codes de sortie $[lrc_2^z + c \cdot 2^{w_1}]$ et $[rrc_2^z + c \cdot 2^{w_1}]$ correspondent au comparateur z du deuxième étage et les codes de sortie du CAN $[lrc_2^w + c \cdot 2^{w_1}]$ et $[rrc_2^w + c \cdot 2^{w_1}]$ correspondent au comparateur w du deuxième étage.

Il reste à trouver les valeurs Y_{k+1} et $Y_{k+1} + t$. Si la première des deux transitions considérées est une transition naturelle du $j^{ième}$ comparateur du $k^{ième}$ étage, alors Y_{k+1} est égale au second élément de R_j^k, qui est $f(k+1, comp_j^k, R)$. Sinon s'il s'agit d'une transition forcée due à la transition naturelle du comparateur h dans le $(k-1)^{ième}$ étage, alors Y_{k+1} est égale au troisième élément de R_h^{k-1}, qui est $f(k+2, comp_h^{k-1}, R)$. Et ainsi de suite.

Nous avons implémenté un algorithme qui permet de calculer tous les codes de sortie du CAN correspondants à chacun des comparateurs de chacun des étages en se basant uniquement sur les L_i^k et R_i^k de chaque comparateur. Le principe de l'algorithme et de son implémentation sont exhaustivement expliqué dans le Chapitre 4.

7.5 Résultats expérimentaux

7.5.1 L'expériment

Notre cas d'étude est un CAN pipeline 11 bits, 55 nm de STMicroelectronics. Ce CAN est composé de quatre étages 2.5-bits et un dernier étage 3 bits. Pour ce CAN particulier, nous avons eu accès à la sortie numérique des étages internes avant que la correction numérique soit appliquée (ce qui fait un total de 15 bits). Ainsi, nous avons extrait la sortie numérique des étages internes et nous avons fait le traitement des données dans Matlab. La Figure 7.20 montre une photo de la carte de test et la Figure 7.21 montre une photo de la configuration de test.

Nous avons utilisé une sinusoïde comme stimulus d'entrée. Nous avons fait un test par histogramme classique afin de comparer à notre technique. Le nombre d'échantillons

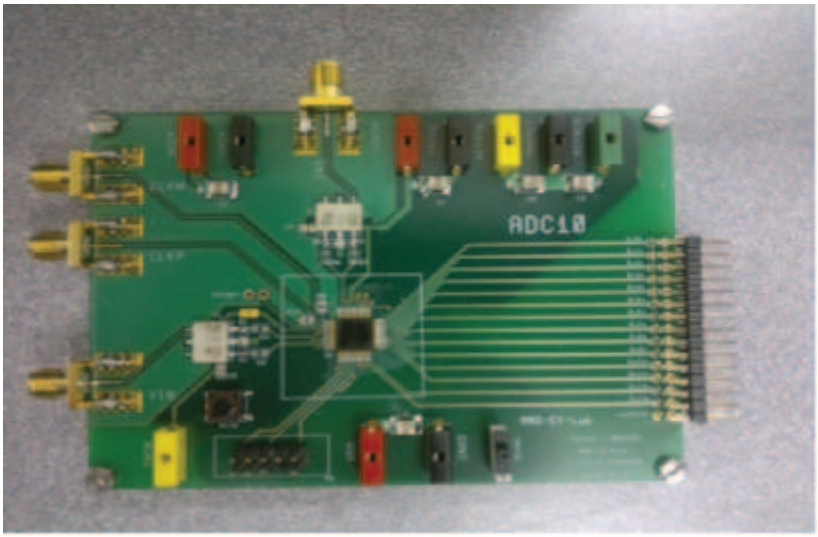


Figure 7.20: La carte de test.

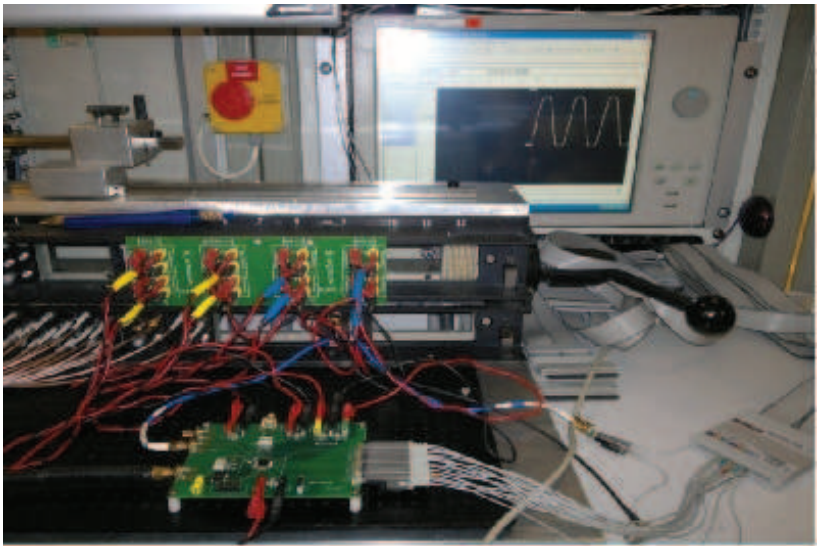


Figure 7.21: L'expériment.

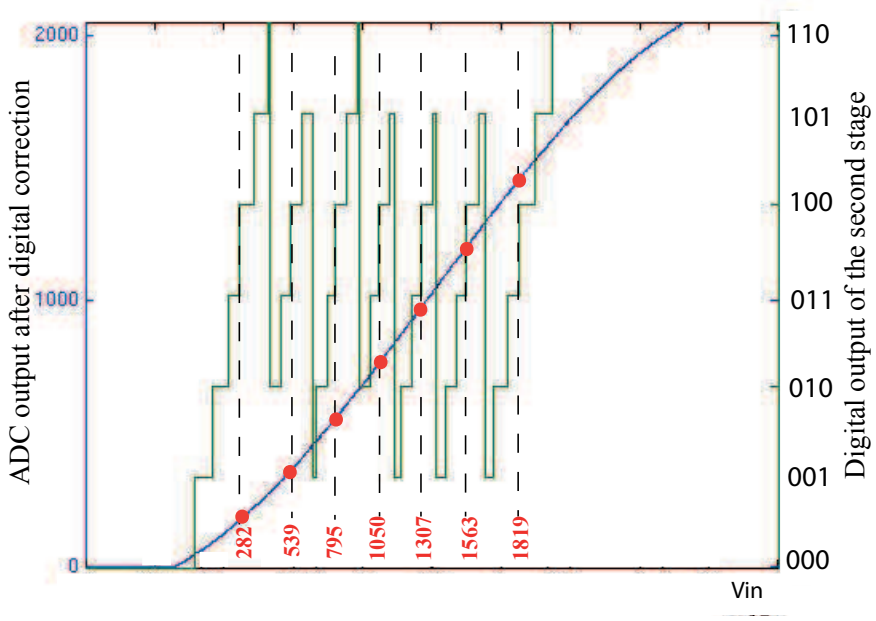


Figure 7.22: Recherche des codes qui correspondent au quatrième comparateur du deuxième étage.

à acquérir pour un test de l'histogramme sinusoïdale dépend de la résolution du CAN et du type d'échantillonnage (cohérent ou aléatoire).

7.5.2 Vérification des principes de base

Tout d'abord, nous avons vérifié la première propriété de la technique qui stipule que les codes liés aux transitions de sortie du CAN qui impliquent le même comparateur ont des NLDs égales. Considérons la Figure 7.22 qui superpose la sortie numérique du CAN sur la sortie numérique du deuxième étage. A titre d'exemple, les points d'intersection entre les lignes verticales en pointillé et la sortie numérique du CAN correspondent aux codes qui impliquent le quatrième comparateur du deuxième étage (passage de 011 à 100). Les codes à droite de ces transitions sont reportés en bas de la Figure 7.22. La première transition correspond au code 282, la deuxième transition correspond au code 539, la troisième transition correspond au code 795, et ainsi de suite.

Ensuite, la NLD de tous les codes a été calculée en utilisant la technique de l'histogramme, comme le montre la Figure 7.24. Les flèches verticales montrent les NLD des codes de la Figure 7.22. Comme on peut le voir, ils sont très proches l'une de l'autre (elles varient entre 0,4278 et 0,4773), ce qui confirme le principe de base de la technique. Ainsi, au lieu de mesurer la NLD de ces sept codes, nous pouvons choisir de mesurer la NLD d'un seul code, puis utiliser cette valeur de NLD pour le reste des codes.

Ensuite, nous avons vérifié un autre principe de la technique qui stipule que les plus grandes erreurs de NLD se produisent autour des transitions de sortie qui impliquent

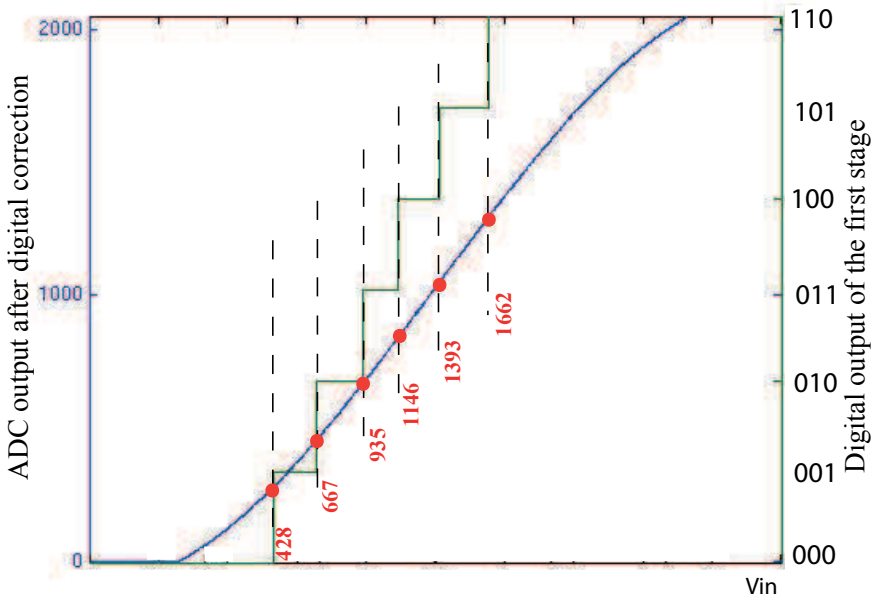


Figure 7.23: Rercherche des codes qui correpondent aux comparateurs du premier étage.

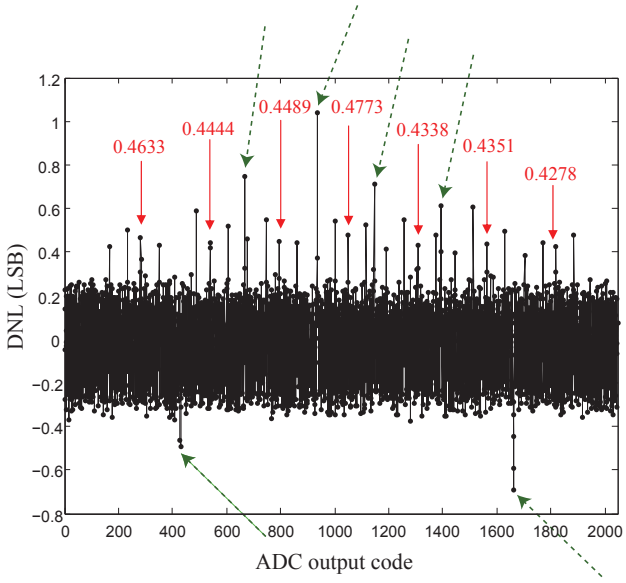


Figure 7.24: NLD obtenue avec la technique standard d'histogramme. Les flèches verticales indiquent les codes qui correspondent au quatrième comparateur du deuxième étage; et les flèches inclinées indiquent les codes qui correspondent aux comparateurs du premier étage.

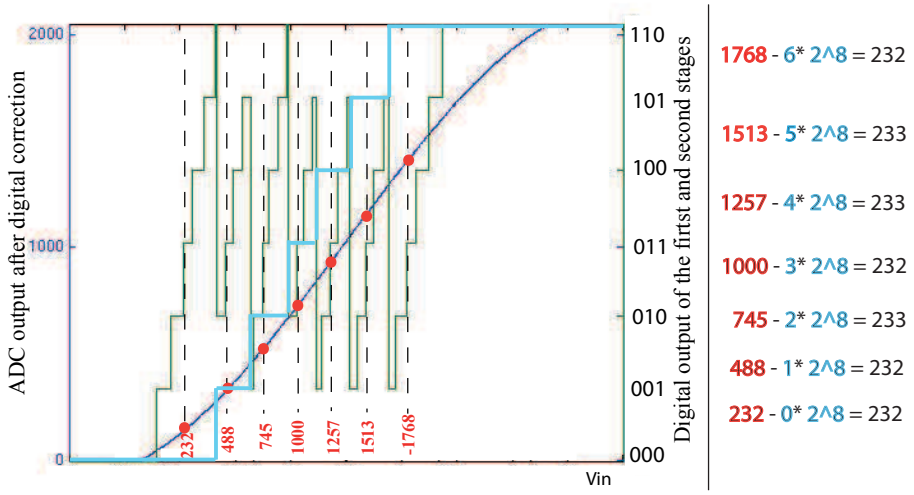


Figure 7.25: Exemple de codes racine.

les comparateurs des étages qui sont vers le début du CAN pipeline. A titre d' exemple, la Figure 7.23 montre la sortie numérique du premier étage superposée sur la sortie numérique du CAN. Les six codes indiqués en bas de la Figure 7.23 sont les codes à droite des transitions du premier étage. Dans la Figure 7.24, ces codes sont indiqués avec les flèches inclinées en pointillés. Comme on le voit, ces codes présentent le maximum et le minimum des erreurs de NLD, ce qui justifie pourquoi nous devrions éviter de sélectionner une transition de sortie du CAN représentative pour un certain comparateur qui implique, en plus de ce comparateur, un autre comparateur dans l'un des étages précédents. Ceci justifie aussi pourquoi nous devons donner la priorité aux premiers étages dans l'étape où nous déduisons les largeurs des codes non mesurées de celles mesurées.

Ensuite, nous avons vérifié la propriété des codes racine que nous avons discuté précédemment. La Figure 7.25 superpose la sortie numérique du CAN sur la sortie numérique du deuxième étage et du premier étage. A titre d'exemple, les points d'intersection entre les lignes verticales en pointillé et la sortie numérique du CAN correspondent aux codes dûs aux transitions qui impliquent le troisième comparateur du deuxième étage (passage de 010 à 011). Les codes à droite de ces transitions sont reportés en bas de la Figure 7.25. La première transition correspond au code 232, la deuxième transition correspond au code 488, la troisième transition correspond au code 745 , etc.

Prenons ces codes (232, 488, 745, 1000, 1257, 1513 et 1768) et enlevons la contribution du premier étage multipliée par le poids de cet étage. Le poids du premier étage de ce CAN est égal à 2^8. Nous obtenons :

$(1768 - 6 * 2^8 = 232)$
$(1513 - 5 * 2^8 = 233)$
$(1257 - 4 * 2^8 = 233)$

$(1000 - 3 * 2^8 = 232)$
$(745 - 2 * 2^8 = 233)$
$(488 - 1 * 2^8 = 232)$
$(232 - 0 * 2^8 = 232)$

Comme prévu, on obtient le même code racine et à cause de la présence du bruit cette valeur alterne entre 232 et 233. 232 semble être la plus fréquente et donc elle définit le code racine droit du troisième comparateur du deuxième étage. Cela signifie que les codes qui doivent être regroupés comme ayant les mêmes NLDs ne sont pas (232, 488, 745, 1000, 1257, 1513 et 1768) comme il aurait été proposé sans utiliser la méthode d'annulation du bruit, mais plutôt les codes (232, 488 , 744, 1000, 1256, 1512 et 1768). Ces derniers sont les codes qui correspondent à ce comparateur recalculés à partir du même code racine (232) en rajoutant les différentes valeurs de la contribution du premier étage.

7.5.3 Résultats expérimentaux

Après avoir vérifié les principes de la technique, nous avons calculé la NLD et la NLI à partir d'un ensemble réduit de codes. Nous avons considéré :

- 8 codes autour de chacune des six transitions de sortie du CAN représentant les six comparateurs du premier étage,
- 6 codes autour de chacune des six transitions de sortie du CAN représentant les six comparateurs du deuxième étage,
- 4 codes autour de chacune des six transitions de sortie du CAN représentant les six comparateurs du troisième étage,
- deux codes autour de chacune des six transitions de sortie du CAN représentant les six comparateurs des quatrième et cinquième étages

Au total, nous avons considéré $8 \times 6 + 6 \times 6 + 4 \times 6 + 2 \times 6 + 2 \times 6 = 132$ codes parmi les 2046 codes d'un CAN 11-bit, ce qui représente uniquement 6% de codes.

La NLD et NLI obtenues en utilisant la technique de l'histogramme sont présentées sur la Figure 7.26(a) et la Figure 7.27(a), respectivement. La Figure 7.26(b) et la Figure 7.27(b) montrent la NLD et la NLI obtenues en utilisant la technique de test à code réduit, dans laquelle la correspondance entre les codes de sortie du CAN et les comparateurs exercés est obtenu en lisant directement les valeurs des sorties numériques des étages à gauche et à droite de la transition. La Figure 7.26(c) et la Figure 7.27(c) montrent la NLD et la NLI obtenues en utilisant la technique de test à code réduit qui prend en compte la méthode d'élimination du bruit. On voit bien que l'estimation de NLD et NLI est très bonne, avec une nette amélioration lors de l'utilisation de la méthode d'élimination du bruit. Ces résultats sont amplement commentés dans le Chapitre 5.

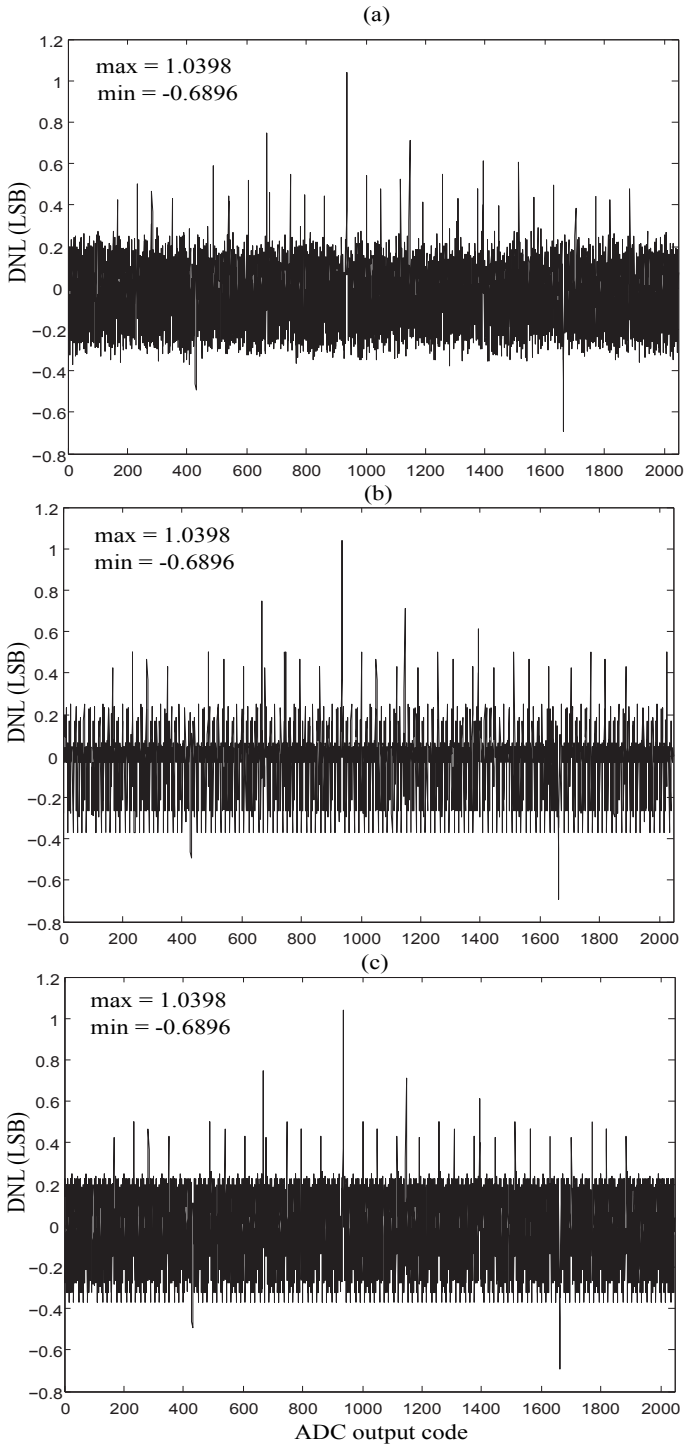


Figure 7.26: NLD obtenue par : (a) la méthode standard d'histogramme; (b) la technique de test à code réduit sans élimination de l'effet du bruit; and (c) la technique de test à code réduit avec l'élimination de l'effet du bruit.

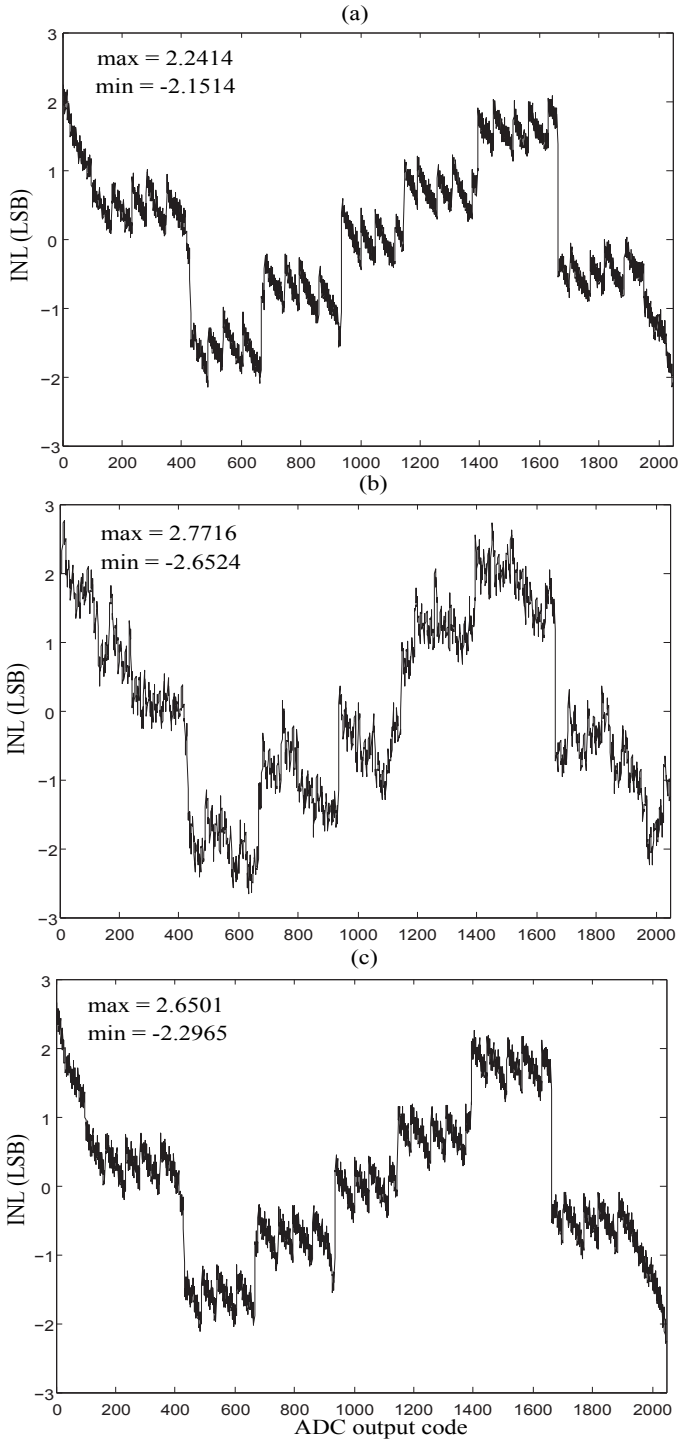


Figure 7.27: NLI obtenue par : (a) la méthode standard d'histogramme; (b) la technique de test à code réduit sans élimination de l'effet du bruit; and (c) la technique de test à code réduit avec l'élimination de l'effet du bruit.

7.6 Résumé des contributions et perspectives

7.6.1 Résumé des contributions

Dans cette thèse j'ai proposé des techniques très efficaces afin d'appliquer le test de linéarité à code réduit pour les CANs de type pipeline. La technique du test à code réduit est basée sur le fait que les codes autour des transitions de sortie du CAN qui sont dues au même comparateur ont des largeurs égales. Ainsi, pour extraire la caractéristique de transfert complète, il suffit de considérer un sous-ensemble représentatif de transitions de sortie qui couvre tous les comparateurs dans tous les étages.

La technique proposée est basée sur l'observation des transitions à la sortie numérique des étages. En effet, à chaque fois que le seuil d'un comparateur est franchi, cela produit forcément une transition dans la sortie numérique de l'étage, et le changement spécifique d'un code à l'autre nous donne l'information sur lequel des comparateurs a été exercé.

Le fait que la technique proposée repose sur l'observation des transitions de la sortie numérique de l'étage la rend sensible au bruit. Ce problème a été résolu en identifiant une autre propriété des CANs pipeline qui a pu être utilisée pour éliminer les erreurs d'estimation qui se produisent en raison de la présence de bruit.

Finalement, les techniques ont été validées expérimentalement sur un CAN 11-bit, 55nm de STMicroelectronics. Les principes de base ont été vérifiées et l'efficacité d'estimation de NLD et NLI a été démontrée.

7.6.2 Perspectives

A part les CANs de type flash et les CANs sigma-delta, tous les autres types de CANs réalisent la conversion en plusieurs étapes. Ainsi, si leur architecture est étudiée en détail, ils offrent tous la possibilité d'appliquer des tests de linéarité à code réduit. Les principes présentés dans cette thèse peuvent être généralisés pour d'autres architectures en tenant en compte leurs particularités.

Aussi, les techniques de test de linéarité à code réduit ouvrent la voie pour le développement de techniques efficaces de test intégré (BIST), car un problème majeur des techniques de test intégré est la nécessité de considérer le nombre très élevé de codes des CANs à haute résolution.

Bibliography

[1] De Vries, T. Zwemstra, E. M. J. G. Bruls, and P. P. L. Regtien. Built-in self-test methodology for A/D converters. In *Proc. IEEE European Design and Test Conference*, pages 353–358, 1997.

[2] Y. C. Wen and K. J. Lee. An on chip ADC test structure. In *Proc. IEEE Design, Automation and Test in Europe Conference*, pages 221–225, 2000.

[3] Y. C. Wen. A BIST scheme for testing analog-to-digital converters with digital response analyses. In *Proc. IEEE VLSI Test Symposium*, pages 221–225, 2005.

[4] F. Adamo, F. Attivissimo, N. Giaquinto, and M. Savino. FFT test of A/D converters to determine the integral nonlinearity. *IEEE Transactions on Instrumentation and Measurement*, 51(5):1050–1055, 2002.

[5] C. Wegener and M.P. Kennedy. Linear model-based testing of ADC nonlinearities. *IEEE Transactions on Circuits and Systems*, 51(1):213–217, 2004.

[6] Z. Yu and D. Chen. Algorithm for dramatically improved efficiency in ADC linearity test. In *Proc. International Test Conference*, pages 1–10, 2012.

[7] J. Lin, T. Kung, and S. Chang. A reduced code linearity test method for pipelined AD converters. In *Proc. IEEE Asian Test Symposium*, pages 111–116, 2008.

[8] F. Poehl, F. Demmerle, J. Alt, and H. Obermeir. Production test challenges for highly integrated mobile phone SOCs–a case study. In *Proc. IEEE European Test Symposium*, pages 17–22, 2010.

[9] IEEE standard for terminology and test methods for analog-to-digital converters.

[10] M. Mahoney. *DSP-Based Testing of Analog and Mixed-Signal Circuits*. Springer, 1987.

[11] Mark Burns and Gordon W. Roberts. *An Introduction to Mixed-Signal IC Test and Measurement*. Oxford University Press 2001, 2001.

[12] IEEE standard for digitizing waveform recorders.

[13] S. Max. Optimum measurement of ADC code transitions using a feedback loop. In *Instrumentation and Measurement Technology Conference, 1999. IMTC/99. Proceedings of the 16th IEEE*, volume 3, pages 1415–1420, 1999.

[14] S. Max. Testing high speed high accuracy analog to digital converters embedded in systems on a chip. In *International Test Conference*, pages 763–771, 1999.

[15] A. Sabatini and P. Carbone. Measurement of static ADC nonlinearities using the servo-loop method. *IEEE Transactions on Instrumentation and Measurement*, 55(5):1772–1777, 2004.

[16] K. Arabi and B. Kaminska. Efficient and accurate testing of analog-to-digital converters using oscillation-test method. In *European Design and Test Conference Proceedings*, pages 348–352, 1997.

[17] Z. Zhao and A. Ivanov. Embedded servo loop for ADC linearity testing. *Microelectronics Journal*, 33:773–780, 2002.

[18] H. Xing, H. Jiang, D. Chen, and R. L. Geiger. High-resolution ADC linearity testing using a fully digital-compatible BIST strategy. *IEEE Transactions on Instrumentation and Measurement*, 58(8):2697–2705, 2009.

[19] B. Provost and E. Sanchez-Sinencio. Auto-calibrating analog timer for on-chip testing. In *Proc. International Test Conference*, pages 541–548, 2010.

[20] E.S. Erdogan and S. Ozev. An ADC-BIST scheme using sequential code analysis. In *Proc. Design Automation Test in Europe Conference Exhibition*, pages 1–6, 2007.

[21] F. Azais, S. Bernard, Y. Bertrand, X. Michel, and M. Renovell. A low-cost adaptive ramp generator for analog BIST applications. In *Proc. IEEE VLSI Test Symposium*, pages 266–271, 2001.

[22] W.-T. Lee, Y.-Z. Liao, J.-C. Hsu, Y.-S. Hwang, and J.-J. Chen. A high precision ramp generator for low cost ADC test. In *International Conference on Solid-State and Integrated-Circuit Technology*, pages 2103–2106, 2008.

[23] B. Provost and E. Sanchez-Sinencio. On-chip ramp generators for mixed-signal BIST and ADC self-test. *IEEE Journal of Solid-State Circuits*, 38(2):263–273, 2003.

[24] B. Provost and E. Sanchez-Sinencio. A practical self-calibration scheme implementation for pipeline ADC. *IEEE Transactions on Instrumentation and Measurement*, 53(2):448–456, 2004.

[25] F. Alegria, P. Arpaia, P. Daponte, and A. Serra. An ADC histogram test based on small-amplitude waves. *Measurement*, 31(4):271–279, 2001.

[26] F. Alegria, P. Arpaia, A.M. Cruz Serra, and P. Daponte. Performance analysis of an ADC histogram test using small triangular waves. *IEEE Transactions on Instrumentation and Measurement*, 51(4):723–729, 2002.

[27] S. Bernard, F. Azais, Y. Bertrand, and M. Renovell. A high accuracy triangle-wave signal generator for on-chip ADC testing. In *Proc. IEEE European Test Workshop*, pages 89–94, 2002.

[28] J. Duan, D. Chen, and R. Geiger. Phase control of triangular stimulus generator for ADC BIST. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 1935–1938, 2010.

[29] L. Jin, K. Parthasarathy, T. Kuyel, D. Chen, and R.L. Geiger. Accurate testing of analog-to-digital converters using low linearity signals with stimulus error identification and removal. *IEEE Transactions on Instrumentation and Measurement*, 54(3):1188–1199, 2005.

[30] L. Jin, K. Parthasarathy, T. Kuyel, R. Geiger, and D. Chen. High-performance ADC linearity test using low-precision signals in non stationary environments. In *Proc. International Test Conference*, pages 1191–1201, 2005.

[31] S. Bernard, M. Comte, F. Azaïs, Y. Bertrand, and M. Renovell. Efficiency of spectral-based ADC test flows to detect static errors. *Journal of Electronic Testing: Theory and Applications*, 20(3):257–267, 2004.

[32] Z. Yu, D. Chen, R. Geiger, and Y. Papantonopoulos. Pipeline ADC linearity testing with dramatically reduced data capture time. In *Proc. IEEE International Symposium on Circuits and Systems*, pages 792–795, 2005.

[33] S. Goyal and A. Chatterjee. Test time reduction of successive approximation register A/D converter by selective code measurement. In *Proc. International Test Conference*, pages 225–233, 2005.

[34] S. Goyal and A. Chatterjee. Linearity testing of A/D converters using selective code measurement. *Journal of Electronic Testing: Theory and Applications*, 24(6):567–576, 2008.

[35] J. Lin, S. Chang, and C. Huang. Design-for-test circuit for the reduced code based linearity test method in pipelined ADCs with digital error correction technique. In *Proc. IEEE Asian Test Symposium*, pages 57–62, 2009.

[36] J. Lin, S. Chang, T. Kung, H.Ting, and C. Huang. Transition-code based linearity test method for pipelined ADCs with digital error correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(12):2158–2169, 2010.

[37] F. Maloberti. *Data Converters*. IEEE Computer Society Press, 2007.

[38] R. de Plassche. *CMOS Integrated Analog-to-Digital and Digital-to-Analog Converters*. Kluwer Academic Publishers, 2003.

[39] D. A. Johns and K. Martin. *Analog integrated circuit design*. John Wiley and Sons, 1996.

[40] S. Sutarja and P.R. Gray. A pipelined 13-bit 250-ks/s 5-V analog-to-digital converter. *IEEE Journal of Solid-State Circuits*, 23(6):1316–1323, 1988.

[41] T. Kuyel and H. Bilhan. Relating linearity test results to design flaws of pipelined analog to digital converters. In *International Test Conference Proceedings*, pages 772–779, 1999.

[42] M. Mayes and S. Chin. A 200 mw, 1 msample/s, 16-b pipelined A/D converter with on-chip 32-b microcontroller. *IEEE Journal of Solid-State Circuits*, 31(12):1862–1872, 1996.

[43] S.H. Lewis and P.R. Gray. A pipelined 5-Msample/s 9-bit analog-to-digital converter. *IEEE Journal of Solid-State Circuits*, 22(6):954–961, 1987.

[44] S.H. Lewis, H.S. Fetterman, G.F. Gross, R. Ramachandran, and T. R. Viswanathan. A 10-b 20-Msample/s analog-to-digital converter. *IEEE Journal of Solid-State Circuits*, 27(3):351–358, 1992.

[45] Y. Xiumei, W. Qi, X. Lai, and Y. Huazhong. A low power 12-b 40-MS/s pipeline ADC. *Journal of Semiconductors*, 31(3):1–6, 2010.

[46] A. Laraba, H. G. Stratigopoulos, S. Mir, H. Naudet, and C. Forel. Enhanced reduced code linearity test technique for multi-bit/stage pipeline ADCs. In *Proc. IEEE European Test Symposium*, pages 50–55, 2012.

[47] A. Laraba, H. G. Stratigopoulos, S. Mir, H. Naudet, and G. Bret. Reduced code linearity testing of pipeline ADCs in the presence of noise. In *Proc. IEEE VLSI Test Symposium*, pages 1–6, 2013.

[48] S. Bernard. *Test Intégré pour Convertisseurs Analogique/Numérique, PhD thesis in Systèmes Automatiques et Microélectroniques (Lirmm).* 2001.

[49] F. Kuttner. A 1.2v 10b 20msample/s non-binary successive approximation ADC in 0.13/spl mu/m CMOS. In *IEEE International solid-State Circuits Conference*, pages 176–177, 2002.

Publications

Peer-reviewed International Journals

[1] **Asma Laraba**, Haralampos-G. Stratigopoulos, Salvador Mir, Hervé Naudet and Gerard Bret. Reduced Code Linearity Testing of Pipeline ADCs. *IEEE Design and Test of Computers*, 2013 (accepted, to appear, pre-edit version on IEEE Xplore).

Peer-reviewed International Conferences

[2] **Asma Laraba**, Haralampos-G. Stratigopoulos, Salvador Mir, Hervé Naudet and Gerard Bret. Reduced code linearity testing of pipeline ADCs in the presence of noise. *IEEE VLSI Test Symposium (VTS)*, Berkeley, California, USA, April-May 2013, pp. 1-6 (**Best Paper Award Candidate**).

[3] **Asma Laraba**, Haralampos-G. Stratigopoulos, Salvador Mir, Hervé Naudet and Christophe Forel. Enhanced reduced code linearity test technique for multi-bit/stage pipeline ADCs. *IEEE European Test Symposium (ETS)*, Annecy, France, May 2012, pp. 50-55 (**Best Paper Award**).

Peer-reviewed National Conferences

[4] **Asma Laraba**, Matthieu Dubois, Haralampos-G. Stratigopoulos, Salvador Mir. Evaluation de la technique de test basée sur la mesure d'un nombre réduit de codes pour les convertisseurs analogique-numerique de type pipeline. *Journées Nationales du Réseau Doctoral en Micro-électronique (JNRDM)*, Paris, France, May 2011.

Other presentations

[5] **Asma Laraba**. Selective Code Testing of Pipeline ADCs. *South European Test Seminar*, Sauze d'Oulx, Italy, 2012.

[6] **Asma Laraba**. Design-For-Test of Pipeline Analog-to-Digital Converters. *PhD Forum, Design Automation and Test in Europe*, Grenoble, France, 2013.

[7] **Asma Laraba**. Conception en Vue du Test des Convertisseurs Analogique-Numerique de type Pipeline. *Journées Semba*, Valence, France, 2010.

Design-For-Test of pipeline Analog-to-Digital Converters

Abstract: Differential Non Linearity (DNL) and Integral Non Linearity (INL) are the two main static performances of Analog to-Digital Converters (ADCs) typically measured during production testing. These two performances reflect the deviation of the transfer curve of the ADC from its ideal form. In a classic testing scheme, a saturated sine-wave or ramp is applied to the ADC and the number of occurrences of each code is obtained to construct the histogram from which DNL and INL can be readily calculated. This standard approach requires the collection of a large volume of data because each code needs to be traversed many times to average noise. Furthermore, the volume of data increases exponentially with the resolution of the ADC under test. According to recently published data, testing the mixed-signal functions (e.g. data converters and phase locked loops) of a System-on-Chip (SoC) contributes to more than 30% of the total test time, although mixed-signal circuits occupy a small fraction of the SoC area that typically does not exceed 5%. Thus, reducing test time for ADCs is an area of industry focus and innovation. Pipeline ADCs offer a good compromise between speed, resolution, and power consumption. They are well-suited for a variety of applications and are typically present in SoCs intended for video applications. By virtue of their operation, pipeline ADCs have groups of output codes which have the same width. Thus, instead of considering all the codes in the testing procedure, we can consider measuring only one code out of each group, thus reducing significantly the static test time. In this work, a technique for efficiently applying reduced code testing on pipeline ADCs is proposed. It exploits two main properties of the pipeline ADC architecture and allows obtaining an accurate estimation of the static performances. The technique is validated on an experimental 11-bit, 55nm pipeline ADC from STMicroelectronics, resulting in estimated DNL and INL that are practically indistinguishable from DNL and INL that are obtained with the standard histogram technique, while measuring only 6% of the codes.

Keywords: *Design-For-Test, static test, Analog-to-digital Converter testing, pipeline Analog-to-Digital Converters, reduced code linearity testing, histogram testing.*

Conception en vue du test des Convertisseurs Analogique-Numérique de type pipeline

Résumé: La Non-Linéarité-Différentielle (NLD) et la Non-Linéarité-Intégrale (NLI) sont les performances statiques les plus importantes des Convertisseurs Analogique-Numérique (CAN) qui sont mesurées lors d'un test de production. Ces deux performances indiquent la déviation de la fonction de transfert du CAN par rapport au cas idéal. Elles sont obtenues en appliquant une rampe ou une sinusoïde lente au CAN et en calculant le nombre d'occurrences de chacun des codes du CAN. Ceci permet la construction de l'histogramme qui permet l'extraction de la NLD et la NLI. Cette approche requiert la collection d'une quantité importante de données puisque chacun des codes doit être traversé plusieurs fois afin de moyenner le bruit et la quantité de données nécessaire augmente exponentiellement avec la résolution du CAN sous test. En effet, malgré que les circuits analogiques et mixtes occupent une surface qui n'excède pas généralement 5% de la surface global d'un System-on-Chip (SoC), leur temps de test représente souvent plus que 30% du temps de test global. Pour cette raison, la réduction du temps de test des CANs est un domaine de recherche qui attire de plus en plus d'attention et qui est en train de prendre de l'ampleur. Les CAN de type pipeline offrent un bon compromis entre la vitesse, la résolution et la consommation. Ils sont convenables pour une variété d'applications et sont typiquement utilisés dans les SoCs destinés à des applications vidéo. En raison de leur façon particulière du traitement du signal d'entrée, les CAN de type pipeline ont des codes de sortie qui ont la même largeur. Par conséquent, au lieu de considérer tous les codes lors du test, il est possible de se limiter à un sous-ensemble, ce qui permet de réduire considérablement le temps de test. Dans ce travail, une technique pour l'application du test à code réduit pour les CANs de type pipeline est proposée. Elle exploite principalement deux propriétés de ce type de CAN et permet d'obtenir une très bonne estimation des performances statiques. La technique est validée expérimentalement sur un CAN 11-bit, 55nm de STMicroelectronics, obtenant une estimation de la NLD et de la NLI pratiquement identiques à la NLD et la NLI obtenues par la méthode classique d'histogramme, en utilisant la mesure de seulement 6% des codes.

Mots clés : *Conception en vue du test, test statique, Convertisseur-Analogique-Numérique, CAN de type pipeline, test par histogramme, test statique à code réduit.*